НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Комп'ютерні системи та мережі»

спеціальності 123 «Комп'ютерна інженерія»

на тему: «Ігровий додаток з генерацією рівнів»

Виконав:
студент IV курсу, групи ІО-62
Сухобрус Олександр Ігорович _____

Керівник:
Асистент
Регіда Павло Генадійович _____

Консультант з нормоконтролю:
Професор, доктор технічних наук
Сімоненко Валерій Павлович _____

Рецензент:
ст. викладач кафедри АУТС _____
Шимкович Володимир Миколайович.

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Сергій СТІРЕНКО

«___»_____20__ р.

ЗАВДАННЯ

на дипломний проєкт студенту

Сухобрусу Олександру Ігоровичу

1. Тема проєкту «Ігровий додаток з генерацією рівнів», керівник проєкту Регіда Павло Генадійович, асистент, затверджені наказом по університету від «07» травня 2020 р. № 1081-с

2. Термін подання студентом проєкту 06 червня 2020 р.

3. Вихідні дані до проєкту: технічне завдання, науково-технічна література

4. Зміст пояснювальної записки: огляд існуючих рішень, вибір і обґрунтування підходів і алгоритмів, розробка і тестування додатку

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо) :

1. Блок-схема алгоритму заповнення матриці кімнат – плакат;
2. Блок-схеми алгоритмів з'єднання сусідніх об'єктів матриці – плакат;
3. Діаграма класів – плакат.

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Сімоненко В.П., професор		

7. Дата видачі завдання 01 вересня 2019 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Затвердження теми роботи	01.09.2019-22.12.2019	
2	Вивчення та аналіз завдання	23.12.2019-22.03.2020	
3	Огляд існуючих додатків та алгоритмів	23.03.2020-01.04.2020	
4	Огляд і вибір інструментів для розробки	02.04.2020-10.04.2020	
5	Розробка алгоритму генерації та формування зображення	11.04.2020-20.04.2020	
6	Розробка додатку та його тестування	21.04.2020-30.04.2020	
7	Оформлення пояснювальної записки	01.05.2020-23.05.2020	
8	Передзахист	24.05.2020-26.05.2020	
9	Захист	15.06.2020-20.06.2020	

Студент Олександр СУХОБРУС

Керівник Павло РЕГІДА

№ рядка	Формат	Позначення	Найменування	Кількість	Примітка
1	А4		Завдання на дипломний проект	2	
2	А4	ІАЛЦ.467800..001 ВП	Відомість дипломного проекту	1	
3	А4	ІАЛЦ.467800..002 ТЗ	Технічне завдання	3	
4	А4	ІАЛЦ.467800..003 ПЗ	Пояснювальна записка	55	
5	А3	ІАЛЦ.467800..004 Д1	Блок-схема алгоритму заповнення матриці кімнат	1	
6	А3	ІАЛЦ.467800..005 Д2	Блок-схема алгоритмів з'єднання сусідніх об'єктів матриці	1	
7	А3	ІАЛЦ.467800.006 Д3	Діаграма класів	1	
8	А3	ІАЛЦ.467800.007 Д4	Лістинг програми	20	

Зм.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.467800.001 ВП		
Розробив		Сухобрус О.І			Ігровий додаток з генерацією рівнів	Літ.	Аркуш 1 / Аркушів 1
Перевір.		Регіда П.Г.					
Н. контр.		Сімоненко В.П.			Відомість дипломного проекту	НТУУ "КПІ ім. Ігоря Сікорського", ФІОТ, ІО-62	
Затверд.							

ТЕХНІЧНЕ ЗАВДАННЯ

до дипломного проєкту

на тему: «Ігровий додаток з генерацією рівнів»

Київ – 2020 року

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання розповсюджується на розробку ігрового додатку з програмною генерацією рівнів.

Область застосування: використання в розважальних цілях на персональних комп’ютерах користувачів.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки служить завдання на виконання розробки ігрового додатку з програмною генерацією рівнів, затвердженою кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний Інститут».

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проекту є розробка ігрового додатку з програмною генерацією рівнів.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелами для розробки служать науково-технічна література з комп’ютерних технологій, публікації в періодичних виданнях, довідники з програмованих логічних інтегральних схем, публікації в Інтернеті за даним питанням.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розроблюваного продукту

- Вивід графічної інформації, що представляє ігрове поле.
- Генерація унікальних ігрових рівнів при кожному запуску.
- Можливість взаємодії з елементами ігрового поля.

5.2. Вимоги до програмного забезпечення

- Операційна система MS Windows XP, MS Windows Vista, MS Windows 7, MS Windows 8/8.1, MS Windows 10
- Підтримка OpenGL 1.2 і вище

5.3. Вимоги до апаратного забезпечення

- Комп’ютер на базі процесору Intel Pentium 2 і вище
- Оперативної пам’яті не менше 128 Мбайт

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проєкту

на тему: «Ігровий додаток з генерацією рівнів»

Київ – 2020 року

ЗМІСТ

ВСТУП

У наш час потреба пересічного користувача в розважальних продуктах є великою як ніколи і буде зростати все більше з кожним роком. Дану закономірність можна прослідкувати по кількості проданих копій ігор в магазинах цифрової дистрибуції. В той же час як затрати на створення якісного ігрового продукту зростають. Великі розробники ігор постійно збільшують штат розробників, що дозволяє витримувати поставлений стандартів. В той же час для розробників, що не мають достатньо ресурсів з'являється необхідність пошуку альтернативних шляхів створення ігор, що могли б зменшити витрати.

Програмна генерація є одним із способів створення рівнів в ігрових додатках, що часто використовується невеликими ігровими студіями і незалежними розробниками. У процесі генерації використовується відносно невелика кількість створених розробником елементів, що дозволяє зменшити затрати на розробку. Завдяки можливостям генерації унікального ігрового оточення, цей підхід дозволяє надати гравцю незабутній ігровий досвід і при цьому створити підстави для повторення ігрової сесії з заново згенерованим для неї оточенням.

Об'єктом дипломної роботи є ігровий рушій ігрового додатку з програмною генерацією рівнів. Дослідження теми зумовлене актуальністю використання програмної генерації для створення ігрового оточення в комп'ютерних іграх.

Предметом даного дослідження є алгоритми генерації рівнів в ігрових додатка.

Метою даної роботи є:

1) Розгляд доцільності використання програмної генерації рівнів в комп'ютерних іграх.
2) Дослідження додатків, що використовують програмну генерацію.

3) Аналіз алгоритмів програмної генерації.
4) Розробка алгоритму програмної генерації рівнів.
5) Створення ігрового додатку з програмною генерацією рівнів.

До другорядних завдань що необхідно виконати для повноцінної роботи додатку є:

1) Розробка принципів формування зображення, що представляє ігрове поле.
2) Розробка методів взаємодії з оточенням.

Дипломна робота включає в себе три розділи. У першому розділі проводиться огляд існуючих додатків з програмною генерацією. Основною метою першого розділу є дослідження використаних в додатках принципів генерації і їх переваги і недоліки. У другому розділі наведено опис існуючих алгоритмів генерації, створеного для власноруч алгоритму генерації, підходу до виведення графічної інформації на екран та алгоритму виявлення зіткнень об’єктів ігрового поля. У третьому і заключному розділі описується структура класів, що використовуються для реалізації поставлених завдань. У кінці роботи приведено загальні висновки, що були зроблені у процесі виконання роботи.

РОЗДІЛ 1
ОГЛЯД ІГРОВИХ ДОДАТКІВ З ГЕНЕРАЦІЄЮ РІВНІВ

1.1 Вступ

Ігрові додатки з програмною генерацією рівнів з'явилися ще в 70 роках минулого століття. Перші спроби представники таких ігор не змогли набути популярності через невдалий вибіл платформи, але в 1980 році після успіху першої гри з генерацією рівнів вони добре укорінилися в ігровій індустрії.

Через складності у використанні програмної генерації у ігрових додатках, що будуються навколо певного сюжету, програмна генерація рівнів в основному зустрічається у таких жанрах як "rougelike" і "sandbox".

Для кращого розуміння принципів і особливостей використання програмної генерації в ігрових додатках буде доцільно розглянути декілька представників, що змогли завоювати визнання і визначити їх основні особливості.

1.2 Rogue

Rouge – комп'ютерна гра 1980 року, що розробили студенти Університету Каліфорнії у Санта-Крузі, Гленн Вічман і Майкл Той для UNIX-подібних операційних систем. Завдяки вільному розповсюдженню по кампусах університету він набула шаленої популярності і в 1984 році була включена до BSD Unix версії 4.2 завдяки чому потрапила на комп'ютери по всьому світу. Величезна популярність гри створила новий жанр комп'ютерних ігор, що отримав назву "rougelike".

Ігровий додаток доступний на платформах:

- Linux
- Amstrad CPC
- Microsoft Windows
- ZX Spectrum

- Mac OS
- Commodore 64
- Amiga
- Atari ST

Ігровий додаток побудований навколо механіки покрокового переміщення по ігровому полю, що представлено поєднанням кімнат, з'єднаних коридорами, і пошуку переходу на наступний рівень з найменшими затратами ресурсів.

У ролі перешкод для гравця виступають ворожі створіння, що мають свій шаблон переміщення по ігровому полю і унікальні особливості і слабкості. Через обмеженість використаного для розробки програмного забезпечення вороги у грі представлені великими літерами латинського алфавіту.

У ролі стимулів для дослідження є корисні ресурси, що випадково генеруються у кімнатах з яких складається підземелля. Окремим елементом ігрового процесу є предмети з невідомими гравцю ефектами, що можуть як бути корисними, так і нашкодити.

Генерація рівнів у додатку виконується на основі сітки розміром три на три комірки. Кожна з комірки сітки підчас генерації може бути обрана для розміщення прямокутної кімнати довільного розміру чи переходу для з'єднання кімнат у сусідніх з ним комірках сітки. Приклад такого рівня представлено на рис.1.1

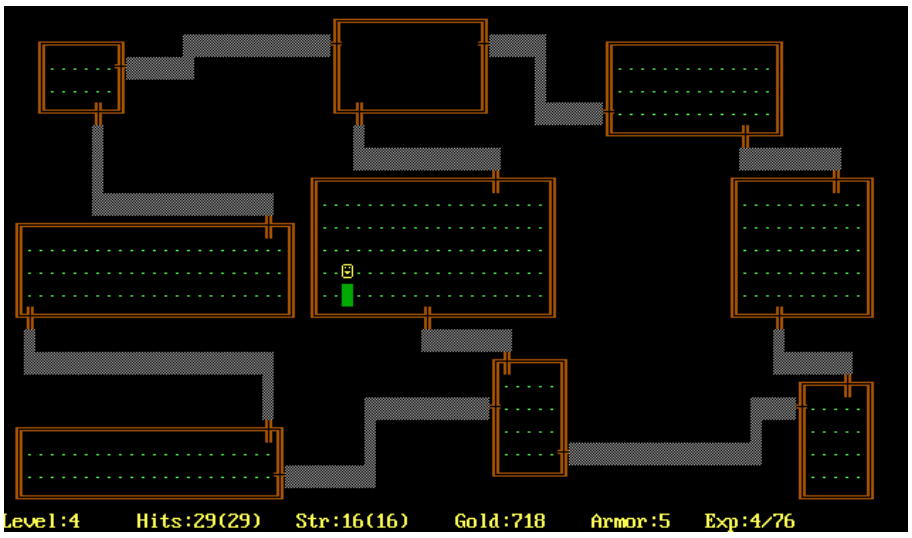


Рис.1.1 Структура рівня комп’ютерної гри Rouge

1.3 The Binding of Isaac

The Binding of Isaac – комп’ютерна гра розроблена Едмунлом Макміланом і Флоріаном Хімслом. Гра вийшла в 2011 році і була створена з використанням Adobe Flash і змогла завоювати любов користувачів, що послугувало підставами для випуску переосмислення в 2014 році, що отримав назву The Binding of Isaac: Rebirth.

Оригінальний додаток доступний на платформах:

- Microsoft Windows
- Mac OS
- Linux

Переосмислена версія доступна на платформах:

- Microsoft Windows
- Mac OS X
- Linux
- PlayStation 4
- PlayStation Vita
- Xbox One
- Wii U
- New Nintendo 3DS

.

- IOS

Ігровий процес додатку побудований навколо певного набору рівнів, що генеруються випадково, але мають окремі особливості генерації. Кожен ігровий рівень складається з набору кімнат. Кожна кімната на рівні може бути одного з наступних типі:

- Звичайна кімната з ворогами
- Звичайна кімната без ворогів
- Кімната з нагородою
- Кімната з фінальним випробування

Кожна з представлених у списку кімнат слугує певній цілі. Так звичайні кімнати з ворогами зачиняються після потрапляння в них гравця і є основним елементом, що перешкоджає гравцю. Кімната з нагородою слугує стимулом для дослідження рівня і може також включати доданкові умови для отримання гравцем бажаного. Звичайні кімнати без ворогів дозволяють розбавити темп гри, а кімната з фінальним випробування слігує переходом на наступний рівень.

Генерація рівню у додатку відбувається шляхом розростання рівня від центра з вибором шаблону кімнати з закованих розробником і наповненню її певними притаманними цьому рівню ворогами. Такий підхід до генерації дозволяє створити унікальний за своєю структурою рівень з контрольованим розробником ігровим процесом в середині кожної окремої кімнати. На рис.1.4 наведено приклад структури такого рівню.

Рис.1.4 Приклад структури рівня у грі The Binding of Isaac

.

Розглянувши переосмислену версію ігрового додатку можна побачити незначні зміни у основних аспектах ігрового процесу, але в той же час система генерації рівнів набула декілька покращень. Основними нововведенням стали кімнати, що займають декілька комірок сітки і відсутність великого нагромадження кімнат, що присутні в оригінальній версії. Приклад такого рівню наведено на рис.1.5.

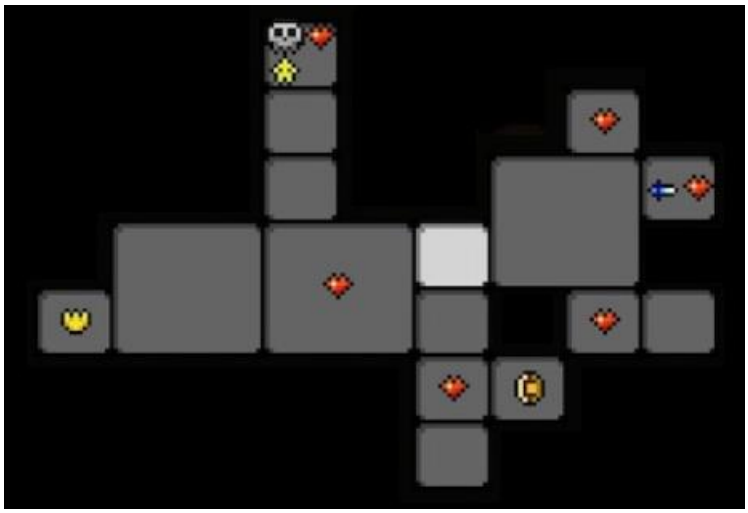


Рис.1.5 Приклад структури рівня у грі The Binding of Isaac: Rebirth

1.4 Spelunky

Spelunky – ігровий додаток, що був розроблений Дериком Йу. Перша версія гри побачила світ 21 грудня 2008 року і мала відкритий програмний код.

Ігровий додаток доступний на платформах:

- Microsoft Windows
- Xbox 360
- PlayStation Vita
- PlayStation 3
- PlayStation 4

Ігрового процесу додатку наслідує основні особливості жанру “платформер” і ставить перед гравцем завдання безпечно дістатись від входу

на рівень, що знаходиться у його верхній частині, до виходу в нижній частині.

Основними перешкодами для гравця у додатку витупають особливості ландшафту. Гравець має слідкувати за висотою спуску і пастками, що знаходяться на карті.

Рівень у додатку представлений прямокутною областю, що не має явного поділу на частини і складається з однакових за розміром блоків. У зв’язку з необхідністю забезпечення можливості гравця дістатися до виходу генератор рівня у додатку завжди прокладає певний шлях між входом і виходом, що не потребує від гравця додаткових засобів.

Генерація рівню[1] у грі відбувається за чітким алгоритмом, що починаючи з верхнього ряду блоків рівня. Першим етапом генерації є розміщення входу в рівень. На кожному наступному кроці алгоритм обирає один із трьох можливих напрямків руху, визначаючи при цьому тип попереднього блоку. По доходженню до нижнього рівня алгоритм може як розмістити вихід, так і продовжити рух вліво чи право до межі рівня. На наступному етапі для кожного блоку обирається вміст відповідно до його типу з заготованих розробником шаблонів. Приклад такого рівню зподілом на блоки з номерами шаблонів наведено на рис.1.6.

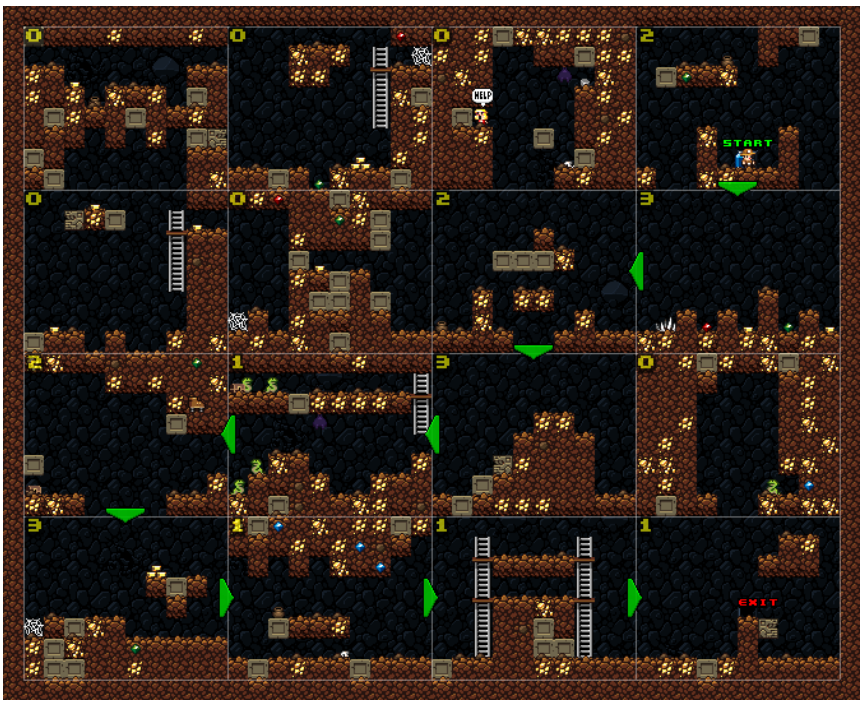


Рис 1.6 Приклад рівня у грі Spelunky

1.5 Enter the Gungeon

Enter the Gungeon – ігровий додаток, що був випущений 5 квітня 2016 року. Розробником гри є студія Dodge Roll з використання ігрового рушія Unity.

Ігровий додаток доступний на платформах:

- Microsoft Windows
- OS X
- Linux
- PlayStation 4
- Xbox One
- Nintendo Switch

Ігровий додаток використовує механіку “шутера” з виглядом зверху і “булетхела”. Основною особливістю додатку є елемент колекціонування зброї при кожному новому ігровому сеансі.

Основними перешкодами для гравця слугують кімнати з ворогами, що появляються в них поступово, в декількох окремих хвилях, що можуть

з’являтися по проходженню певного часу чи після перемоги над попередньою хвилею.

Генерація рівнів у ігровому додатку відбувається з використанням шаблону, що дозволяє створити збалансовану складність шляху до ключових кімнат рівня, але в той же час створює можливість для гравців завчити шлях. Приклад шаблону кімнати наведений на рис.1.7. Випадковим у генерації є розміщення кімнат в просторі і вибір шаблону кімнати. Приклад згенерованого рівня приведено на рис.1.8.

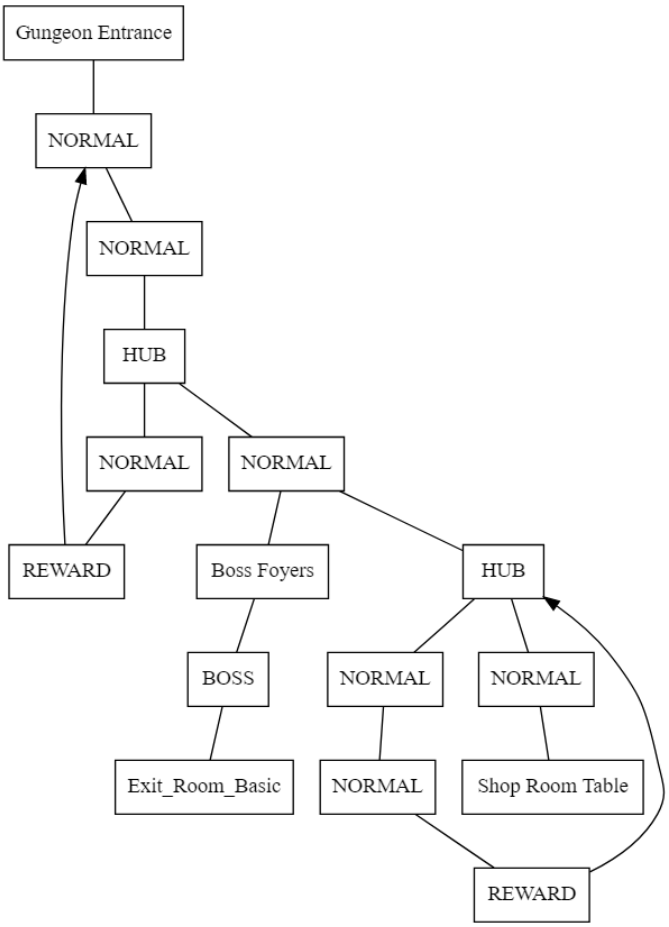


Рис.1.7 Граф структури рівню

.

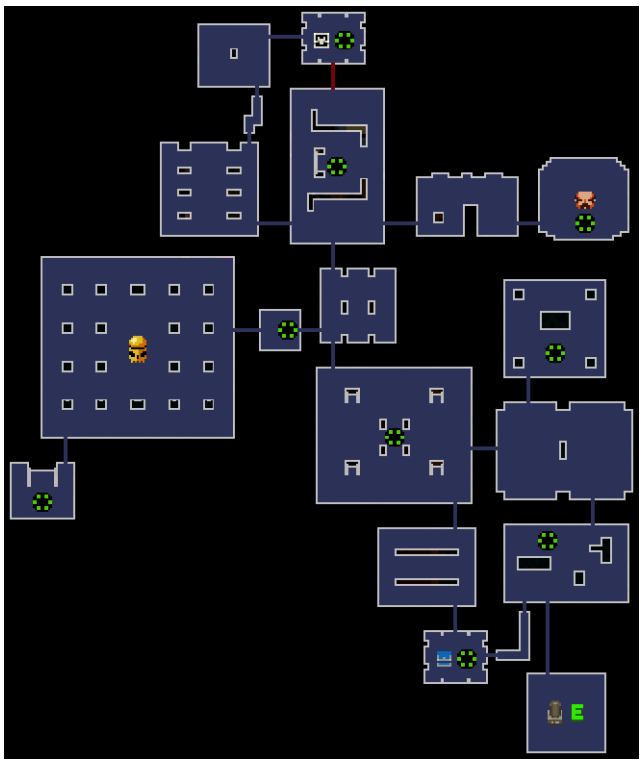


Рис.1.8 Приклад рівню у грі Enter the Gungeon

Іншою особливістю додатку є використання тривимірної графіки для представлення двовимірних об’єктів на екрані. Такий підхід до формування двовимірного зображення дозволяє створити накладення об’єктів ресурсами графічного процесора, а також реалізувати систему освітлення з використанням засобів тривимірної графіки. Приклад реалізації даної ідеї ігровим додатком представленно на рис.1.9. Завдяки зміні кута камери можливо побачити тривимірну природу об’єктів ігрового оточення.

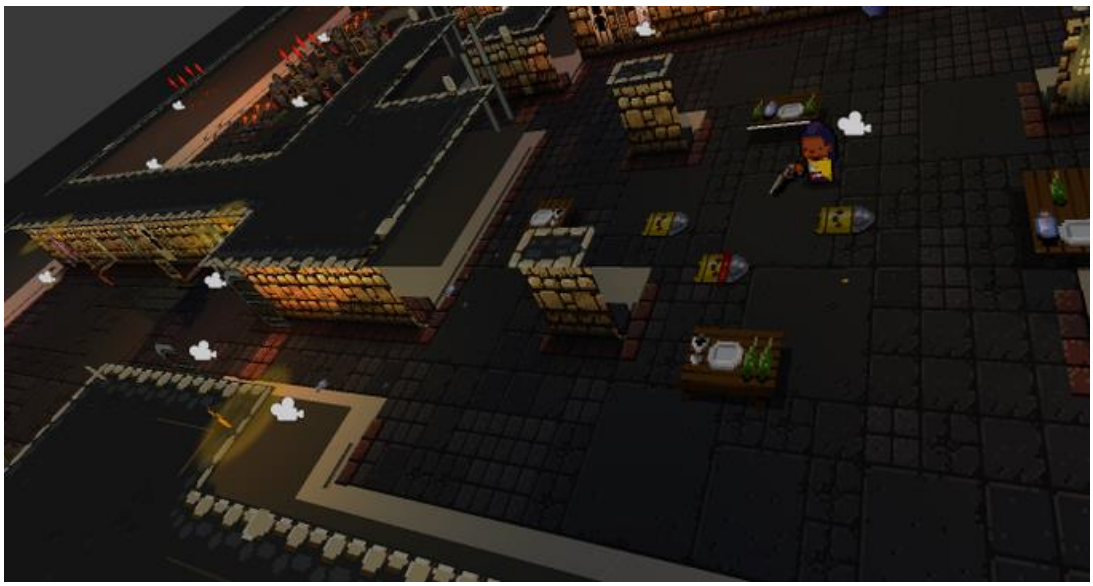


Рис.1.9 Приклад використання тривимірної графіки в ігровому додатку Enter the Gungeon

Якщо детальніше розглянути представлене зображення можна побачити, що для розміщення освітлення, що не має відображатися на верхніх частинах стін виконується їх переміщення вздовж осі, що перпендикулярна площини екрану. Також стає очевидним, що при кінцевій обробці зображення виконується його деформація вздовж вертикальної осі, що дозволяє зберегти правильність пропорцій.

Хоча такий підхід і збільшує апаратні затрати, шляхом ускладнення графічних об’єктів, але в той же час надає значно ширший спектр можливостей у порівнянні із звичайним методом представлення двовимірних об’єктів.

Висновок до розділу 1

У розділі було розглянуто декілька ігрових додатків з генерацією рівнів і розглянуто їх особливості. Кожен із розглянутих додатків має несхожий на інші алгоритм генерації рівнів і ігровий процес. Незважаючи на різноманітність ігрових жанрів елемент генерації рівня органічно вплетений у ігрову логіко.

У процесі дослідження алгоритмів генерації, що використанні в додатках було визначено важливість комбінування рукотворних і згенерованих елементів рівня для досягнення цікавого ігрового процесу.

Незважаючи на різні підходи до реалізації цікавого ігрового процесу, у розглянутих додатках використовується двовимірне ігрове поле і двовимірні об’єкти ігрового оточення. Такий підхід дозволяє зменшити затрати на розробку і зменшує швидкість втрати актуальності графічного оформлення.

В процесі дослідження було виявлено можливість покращення зображення, шляхом використання тривимірного простору для реалізації системи освітлення і перекриття коректного об’єктів ігрового поля у відповідності з їх розміщенням на ньому. Хоча можливість реалізації перекриття об’єктів і можливо реалізувати шляхом сортування об’єктів ігрового поля і відображення їх в правильній послідовності, було вирішено, що використання тривимірного простору є більш якісним підходом.

РОЗДІЛ 2
ОГЛЯД РІШЕНЬ І АЛГОРИТМІВ

2.1 Вступ

Розробка ігрових додатків є складним процесом, що включає в себе багато різноманітних етапів. У даній дипломній роботі основною вимогою до створеного додатку є використання програмної генерації рівнів. З цього можна виділити мінімальний набір складових, що необхідні для робото здатності створеного додатку. До таких складових частин додатку можна віднести:

- Виведення графічної інформації на екран
- Генерація ігрових рівнів
- Визначення взаємодії об’єктів

У даному розділі будуть розглянуті алгоритми, для реалізації кожного з цих завдань.

2.2 Огляд технологічних засобів

Для реалізації основних завдань, що були поставлені перед додатком, що необхідно розробити було обрано бібліотеку SDL і програмний інтерфейс OpenGL.

SDL - це бібліотека для забезпечення абстракції апаратного шару для комп'ютерних мультимедійних компонентів. Розробники програмного забезпечення можуть використовувати бібліотеку для написання високоефективних комп'ютерних ігор та інших мультимедійних додатків.

Створені з допомогою SDL додатки можуть запускатися на таких операційних системах як:

- Android
- iOS
- Linux
- macOS
- Windows

Бібліотека має наступний спектр можливостей:

- Основні – Ініціалізація та вимкнення, змінні конфігурації, обробка помилок, обробка журналу
- Відео – Управління дисплеями та вікнами, поверхневі функції, прискорення візуалізації тощо.
- Вхідні події – Обробка подій, підтримка клавіатури, миші, джойстика та контролера гри
- Примусовий зворотній зв'язок – SDL_haptic.h реалізує підтримку "Примусового зворотного зв'язку"
- Аудіо – SDL_audio.h реалізує управління звуковими пристроями, відтворення та запис
- Потоки – багато потоковість, управління потоками, примітиви для синхронізації потоків, атомарні операції
- Таймери – Підтримка таймеру
- Абстракція файлів – Шляхи файлової системи, абстракція вводу / виводу файлів
- Спільна підтримка об'єктів – Спільне завантаження об'єктів і пошук функцій
- Інформація про платформу та процесор – Виявлення платформи, виявлення функціональних процесорів, порядок байт та заміна байтів, маніпуляція бітами
- Управління живленням – Статус управління живленням
- Додатковий функціонал для платформи

OpenGL – програмний інтерфейс для візуалізації 2D та 3D векторної графіки. Зазвичай його можливості використовуються для досягнення апаратного прискореного візуалізації шляхом взаємодії з графічним процесором.

.

2.3 Графічне представлення об’єктів

Для представлення об’єктів на ігровому полі було обрано двовимірні графічні об’єкти. Для покращення якості отриманого зображення було вирішено реалізувати накладання об’єктів.

Для реалізації накладання об’єктів ігрового поля з використанням правил трьохвимірного простору можливо використати застосувати такі підходи як:

- Сортування об’єктів за порядком накладання
- Використання проекції тривимірного простору

Для даного додатку було обрано метод проекції тривимірного простору. Таке рішення пов’язано з кращою якістю отриманого результату і можливість надалі використовувати ефекти, що будуть розраховуватись з допомогою ресурсів відеоприскорювача.

Для правильного відображення двовимірних об’єктів у трьохвимірному просторі необхідно реалізувати викривлення усіх об’єктів у відповідності до кута нахилу до площини екрану.

У даній роботі було вирішено використати інший підхід, що передбачає розміщення площини екрану паралельно до площини xOy. У такому разі всі об’єкти, що знаходяться в площинах паралельних до площини xOy не потребують жодних перетворень. Усі ж об’єкти, що знаходяться в площині xOz будуть переміщені на площину xOy*, що знаходиться під кутом 45 градусів до площин xOy і xOz. Таке розміщення площини дозволяє розраховувати координати проекції об’єктів з площини xOz на ній збільшенням координати y для точок об’єкта на значення z відповідної точки. Приклад розміщення додаткової осі Oy* наведено на рис.2.1.

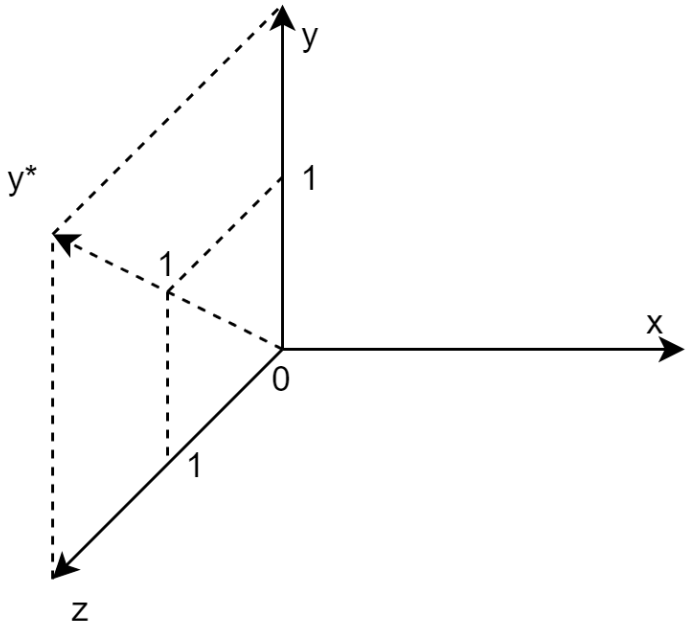


Рис.2.1. Координатний простір з додатковою віссю

Оскільки додаток передбачає від рисовку великої кількості об’єктів, що мають текстури невеликого розширення, то було вирішено використовувати для зберігання текстур об’єктів, текстурні атласи. У такому випадку можливо зменшити загальну кількість запитів до опера до відеоприскорювача, що позитивно впливає на швидкодію додатку.

2.4 Огляд існуючих алгоритмів генерації рівнів

Існує безліч різних методів генерації ігрових рівнів. Для кращого розуміння підходів до генерації можна розглянути такі алгоритми як:

- Алгоритм двійкового поділу простору[3]
- Алгоритм тунелювання[4]
- Клітинкові автомати[5]
- Алгоритм п’яної ходьби[6]

Першим алгоритмом генерації рівнів, що буде розглянутий є алгоритм двійкового поділу простору. Результат роботи цього алгоритму наведений на рис.2.2.

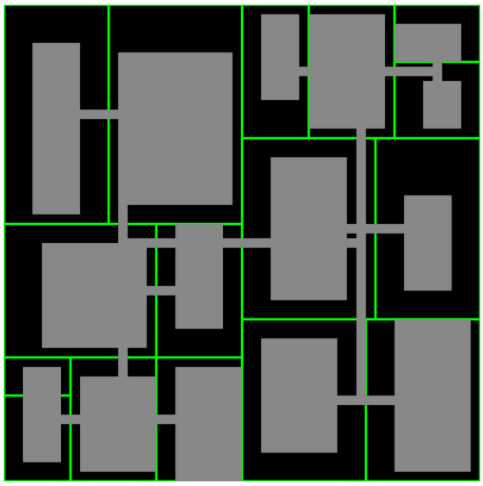


Рис.2.2. Приклад рівню, що згенерований двійковим поділом простору

Алгоритм двійкового поділу простору працює за рахунок створення двійкового дерева, нащадки кожної вершини якого утворюються поділом батьківської на дві частини.

На наступному кроці алгоритму виконується розміщення кімнат в кожній з створених комірок, що не мають нащадків.

Процес з’єднання кімнат коридорами відбувається з допомогою двійкового дерева в якому нащадки однієї вершини мають спільну грань.

Перевагами цього способу генерації є велике різноманіття способів розміщення кімнат, а недоліком є складність включення кімнат створених розробником.

Наступним алгоритмом генерації є алгоритм тунелювання, що дозволяє створювати подібні до рукотворних рівні. Даний ефект досягається завдяки механізмам тунелювання, що мають прокламувати тунелі і розміщувати поруч з ними кімнати у відповідності із заданим розробником набором правил. Приклад такого рівня наведено на рис.2.3.

Рис.2.3. Приклад рівню, що згенерований алгоритмом тунелювання

Одним із недоліків даного методу можна вважати велику кількість тунелів, що не стимулює гравця заходити в кімнати. У зв'язку з цією особливістю кожна кімната має мати стимул для її дослідження.

Наступним алгоритмом генерації є клітинкові автомати. Даний алгоритм генерації використовує як основу матрицю, що випадково заповнена нулями і одиницями. Для формування системи печер метод виконує симуляцію життєдіяльності певних істот, що мають два стани. Кожна з таких істот може змінювати свій стан в залежності від стану сусідів, що дозволяє через декілька проходів алгоритму перетворити випадковиц набір нулів і одиниць матриці на систему печер. Приклад рівню, що згенерований даним методом наведений на рис.2.4

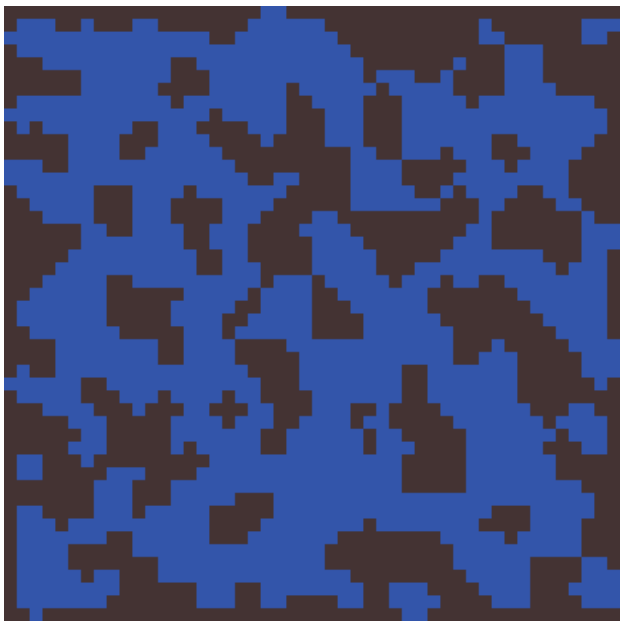


Рис.2.4. Приклад рівню, що згенерований клітинковими автоматами

Хоча даний метод і дозоляє створювати унікальні системи печер, але у зв’язку з відсутністю можливості створення стимулів для гравця, метод необхідно застосовувати у комбінації з іншими.

Останнім методом генерації є метод п’яної ходьби, що дозволяє створювати печери заданого розміру. Алгоритм побудований навколо випадкового переміщення вздовж осей з позначенням не пройдених раніше клітинок до моменту доки на полі не залишиться задана кількість клітинок. Приклад роботи даного алгоритму наведено на рис.2.5.

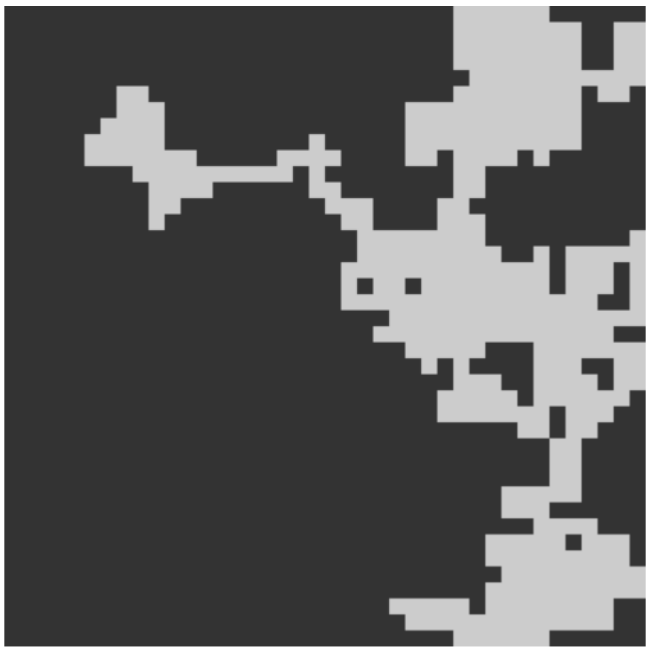


Рис.2.5. Приклад рівню, що згенерований алгоритмом п’яної ходьби

До преваг даного алгоритму випадкової генерації печери є те, що згенерована печера не матиме недоступних місць. Як недолік можна виділити випадкову складність роботи, що не піддається обчисленню.

2.5 Опис використаного алгоритму генерації рівнів

Для генерацій рівнів було розроблено алгоритм, що виконує генерацію на основі сітки з однаковими комірками. Алгоритм генерації розміщує попередньо створенні розробником заготовки кімнат по коміркам сітки, формуючи структуру рівня.

Алгоритм генерації рівнів можна поділити на такі етапи як:

- Визначення комірок сітки, що займають кімнати
- Розрахунок з’єднання кімнат
- Заповнення кімнат плитками
- Розрахунок з’єднання плиток

Переший етап генерації рівня виконує маркування комірок сітки, в яких надалі будуть розміщені кімнати. Для їх маркування передбачено метод визначення кількості зайнятих сусідніх комірок, що реалізується з

допомогою збільшення значення, що записане в кожній сусідній комірці у випадку якщо комірка займається.

У даному варіанті алгоритму враховуються лише комірки, що мають спільні з активною коміркою грані, але для покращення якості згенерованого розміщення кімнат також можливо враховувати кімнати, що мають спільні верни з даною коміркою. Такий варіант потребує додаткових досліджень, оскільки наявність зайнятої комірки, що не має спільної грані є менш вагомою ніж наявність зайнятої комірки зі спільною гранню, і може не враховуватись при відсутності зайнятих комірок, в більш вагомих містях.

На першому кроці алгоритму виконується розміщення початкової кімнати, що і буде центром створеного рівня. На всіх інших виконується випадкове розміщення кімнат в усіх чотирьох можливих напрямках. Виконання алгоритму продовжується до повного перебору всих кімнат із списку активни. Блок-схема алгоритму генерації приведена на рис.2.6.

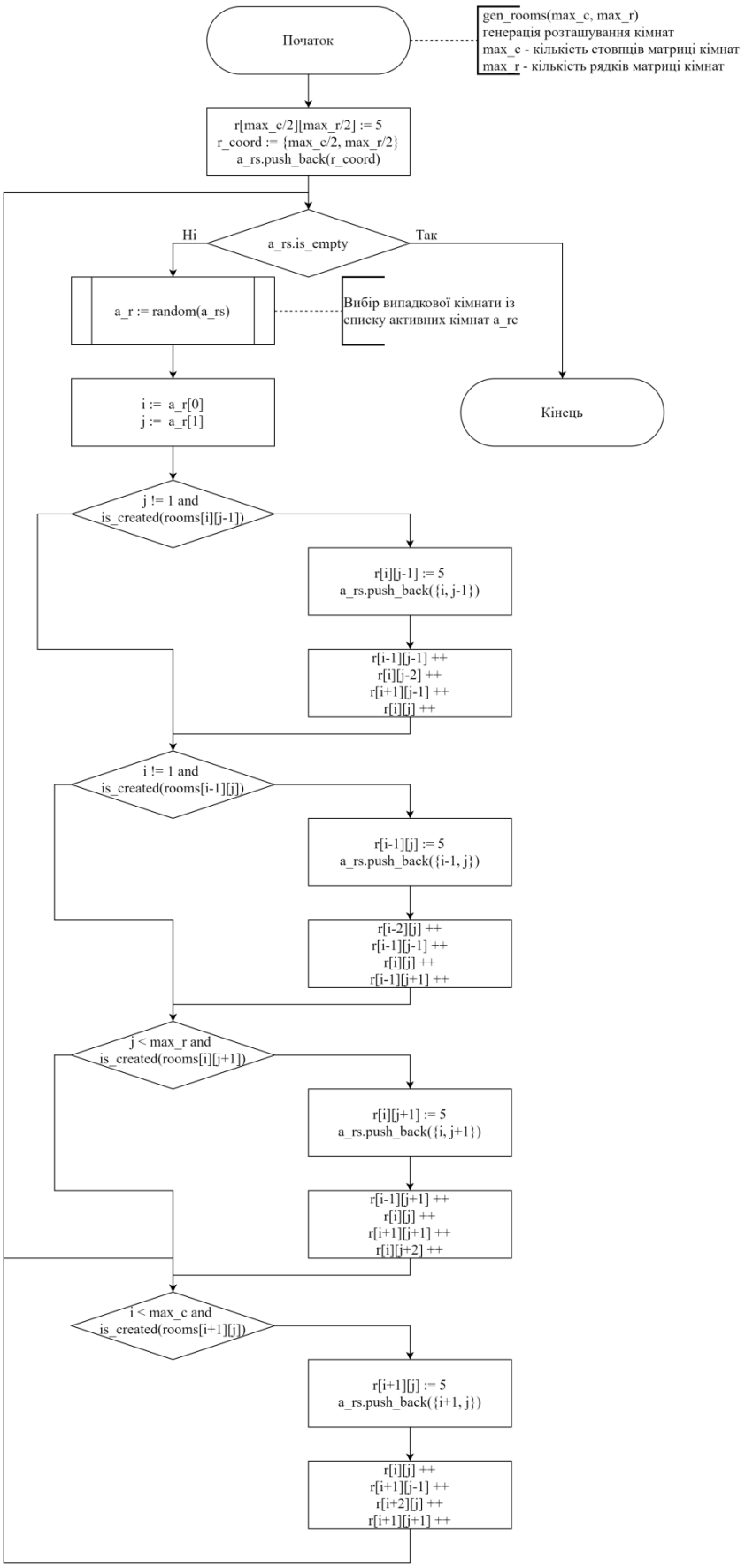


Рис.2.6. Блок-схема алгоритму заповнення матриці кімнат

Приклад роботи першого етапу алгоритму можна побачити на рис.2.7.

```
0 0 0 0 0      0 0 0 0 0      0 0 0 1 0
0 0 1 0 0      0 0 1 1 0      0 0 2 5 1
0 1 5 1 0  =>  0 1 7 5 1  =>  0 1 7 6 1   =>
0 0 1 0 0      0 1 5 2 0      0 1 5 2 0
0 0 0 0 0      0 0 1 0 0      0 0 1 0 0

0 0 0 1 1      0 0 0 1 1      0 0 0 0 0
0 0 2 6 5      0 0 2 6 5      0 0 0 1 1
=> 0 1 7 6 2 => 0 2 7 6 2 =>  0 0 1 1 0
0 1 5 2 0      1 5 7 2 0      0 1 1 0 0
0 0 1 0 0      0 2 5 1 0      0 0 1 0 0
```

Рис.2.7.Приклад заповнення матриці кімнат з нормалізацією на останньому кроці

Наступним кроком алгоритму генерації рівню є з'єднання кімнат для виконання цього кроку алгоритм використовує бітмаскінг[7]. У розробленому алгоритмі виконується розрахунок маски для кімнат з врахуванням чотирьох сусідніх комірок і для плиток з врахуванням восьми сусідніх плиток. Значення бітів, що змінює у масці кожна присутня сусідня комірка наведені на рис.2.8.

	1	
8	x	2
	4	

а)

8	16	1
128	x	32
4	64	2

б)

Рис.2.8.Значення, що з що змінює у масці кожна присутня сусідня комірка

а) для чотирьох бітної маски; б) для восьми бітної маски

На рис.2.9. наведено блок-схему алгоритму з’єднання кімнат, що використовує чотирьох бітну маску.

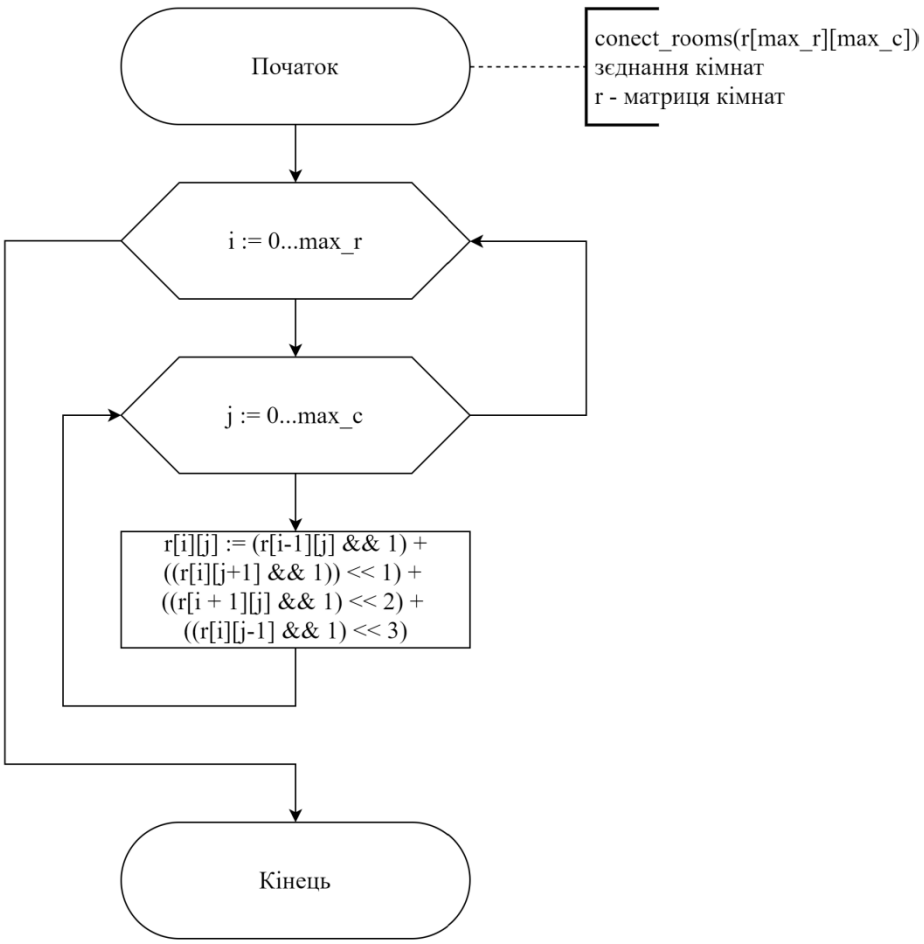


Рис.2.9. Блок-схема алгоритму з’єднання кімнат

Наступним кроком після з’єднання кімнат є заповнення патриці плиток, з яких складається рівень ігрового додатку. Для заповнення плиток використовуються шаблони, що випадково вибираються із набору шаблонів, що відповідає типу кімнати, визначеному на попередньому кроці. Усі шаблони, що використовуються для заповнення плиток зберігаються у файлі і завантажуються при запуску додатку.

Після заповнення типів плиток у відповідності з шаблонами виконується розрахунок значень зєднаних текстур, що будуть використовуватись для відображення кожної плитки. Цей розрахунок відбувається з врахуванням восьми плиток, що оточують плитку, для якої це

значення рахується. На рис.2.10. наведено блок-схему алгоритму з’єднання плиток, що використовує восьми бітну маску.

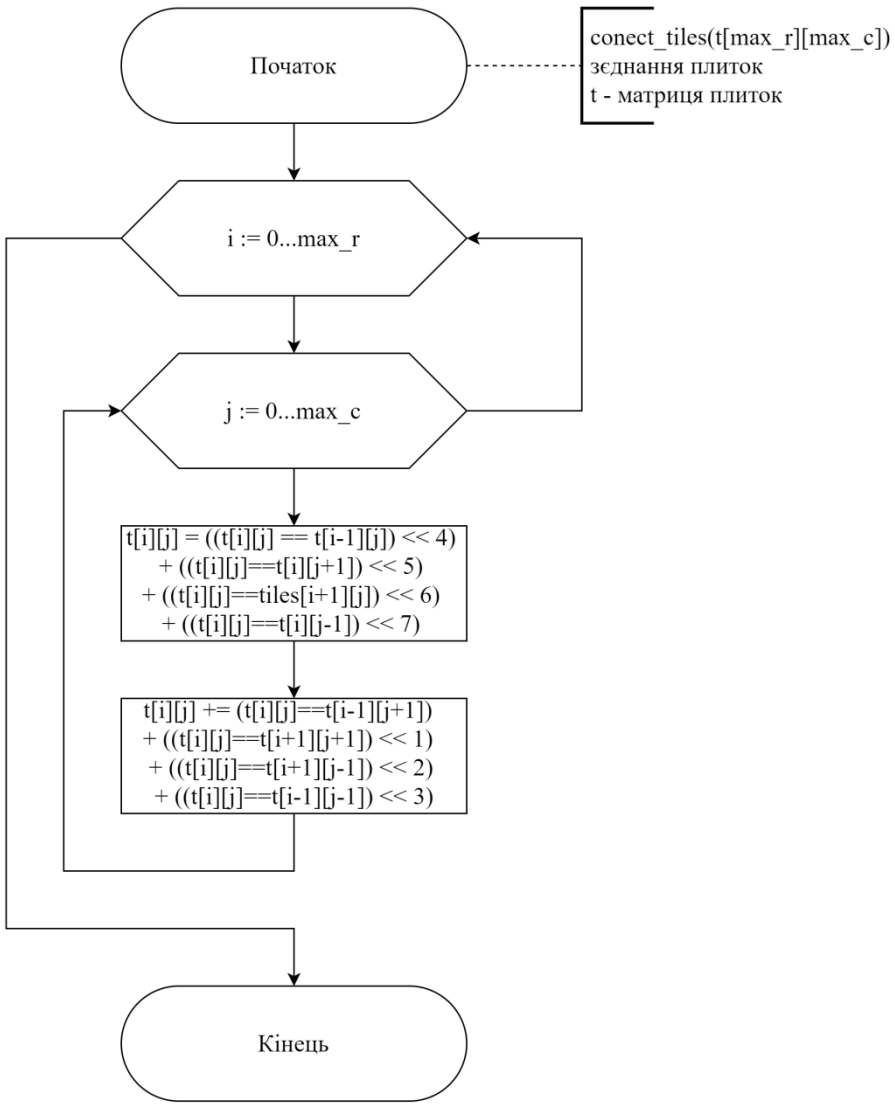


Рис.2.10. Блок-схема алгоритму з’єднання плиток

У результаті обрахунку восьми бітної маски ми отримуємо 256 різних значень, але враховуючи, що значення кутових плиток враховується лише у випадку коли дві центральні плитки, що до неї прилягають, теж присутні, то отримуємо всього 47 унікальних варіантів шаблонів. Для того, щоб використовувати цю особливість у додатку використовується словник, що і переводить відповідне значення у номер шаблону. Відповідність шаблонів до значення номера текстури приведено у таблиці 2.1.

Таблиця 2.1. Відповідність значень шаблонів до значень восьми бітної маски

№ Шаблону	Значення номера текстури	№ Шаблону	Значення номера текстури
0	0 - 15	24	98, 99, 102, 103, 106, 107, 110, 111
1	128 - 143	25	226, 227, 234, 235
2	16 - 31	26	228, 229, 236, 237
3	144 - 151	27	248
4	152 - 159	28	230, 231, 238, 239
5	32 - 47	29	249
6	160 - 175	30	250
7	48, 50, 52, 54, 56, 58, 60, 62	31	112, 116, 120, 124
8	49, 51, 53, 55, 57, 59, 61, 63	32	113, 117, 121, 125
9	176, 178, 180, 182	33	114, 118, 122, 126
10	177, 179, 181, 183	34	115, 119, 123, 127
11	184, 186, 188, 190	35	240,
12	185, 187, 189, 191	36	241

Продовження таблиці 2.1.

№ Шаблону	Значення номера текстури	№ Шаблону	Значення номера текстури
13	64 - 79	37	242
14	192 – 195, 200 - 203	38	243
15	196 – 199, 204 - 207	39	244
16	80 - 95	40	245
17	208 - 211	41	246
18	212 - 215	42	251
19	216 - 219	43	252
20	247	44	253
21	220 - 223	45	254
22	96, 97, 100, 101, 104, 105, 108, 109	46	255
23	224, 225, 232, 233		

Для зберігання усіх 47 варіантів текстури для плитки у текстурному атласі виділяється два рядки довжиною в 24 окремі текстури. Приклад такого набору текстур наведено на рис.2.11.

Рис.2.11. Набір текстур для верхньої частини стіни

2.6 Алгоритм визначення перетину об’єктів

Враховуючи невелику геометричну складність об’єктів ігрового додатку оптимальним і достатнім буде використання алгоритму, що базується на перевірці зіткнень вирівняних по координатних осях обмежуючих прямокутниках[8].

До переваг використання даного алгоритму можна віднести найменшу складність поміж алгоритмів виявлення зіткнень, що зумовлена необхідністю виконання лише чотирьох операції порівняння для виявлення перетину двох об’єктів. Приклад двох об’єктів, що перетинаються показано на рис

Основними недолікоми даного підходу є неможливість точного виявлення перетину двох об’єктів складної форми, що можливо виправити використанням декількох окремих прямокутників для якнайточнішого перекриття об’єкта, і неможливість виконання повороту, що зумовлює необхідність перерахунку відповідних об’єкту обмежуючих прямокутників.

Враховуючи, що виявлення усіх. можливих зіткнень необхідно перебрати усі пари об’єктів, що є на ігровому полі для алгоритмів виявлення зіткнень використовують різноманітні методи оптимізації. В основному такі методи зав’язані на групуванні об’єктів і перевірці на зіткнення об’єктів, що знаходяться в одній групі.

Оскільки в даному додатку передбачено тільки визначення перетинів з статичними об’єктами оточення, що розміщенні згідно сітки, використання жодних алгоритмів оптимізації не є необхідним.

Висновок до розділу 2

У першій частині розділу було поставлено основні вимоги до додатку і обрано набір програмних засобів для їх реалізації. До цих програмних засобів належать бібліотека SDL і програмний інтерфейс OpenGL. Підставою для їх вибору послужила підтримка цими засобами великої кількості платформ і зручність їх використання. Для написання основного коду програми було обрано мову C++, як таку, що сумісна з обраними програмними засобами і та, що надає можливість використання парадигми об’єктно орієнтованого програмування.

У другій частині розділу було описано принцип представлення об’єктів на екрані і описано переваги цього підходу.

У третій частині розділу розглянуто декілька існуючих алгоритмів генерації рівнів і описано запропонований підхід генерації рівнів з центру сітки.

У четвертій частині розділу описано алгоритм визначення перетинів об’єктів ігрового поля і описано доцільність його використання.

РОЗДІЛ 3

РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1 Опис структури програми

Ігровий додаток, що був розроблений в даній дипломній роботі складається з таких класів як:

- Game
- LevelMap
- Texture
- TextureCoord
- Rectangle2_5D
- GrObject
- Tile
- Animation
- Entity

Кожен з цих класі відповідає за свою частину функцій створеного додатку. Спрощена діаграма класів зображена на рис.3.1.

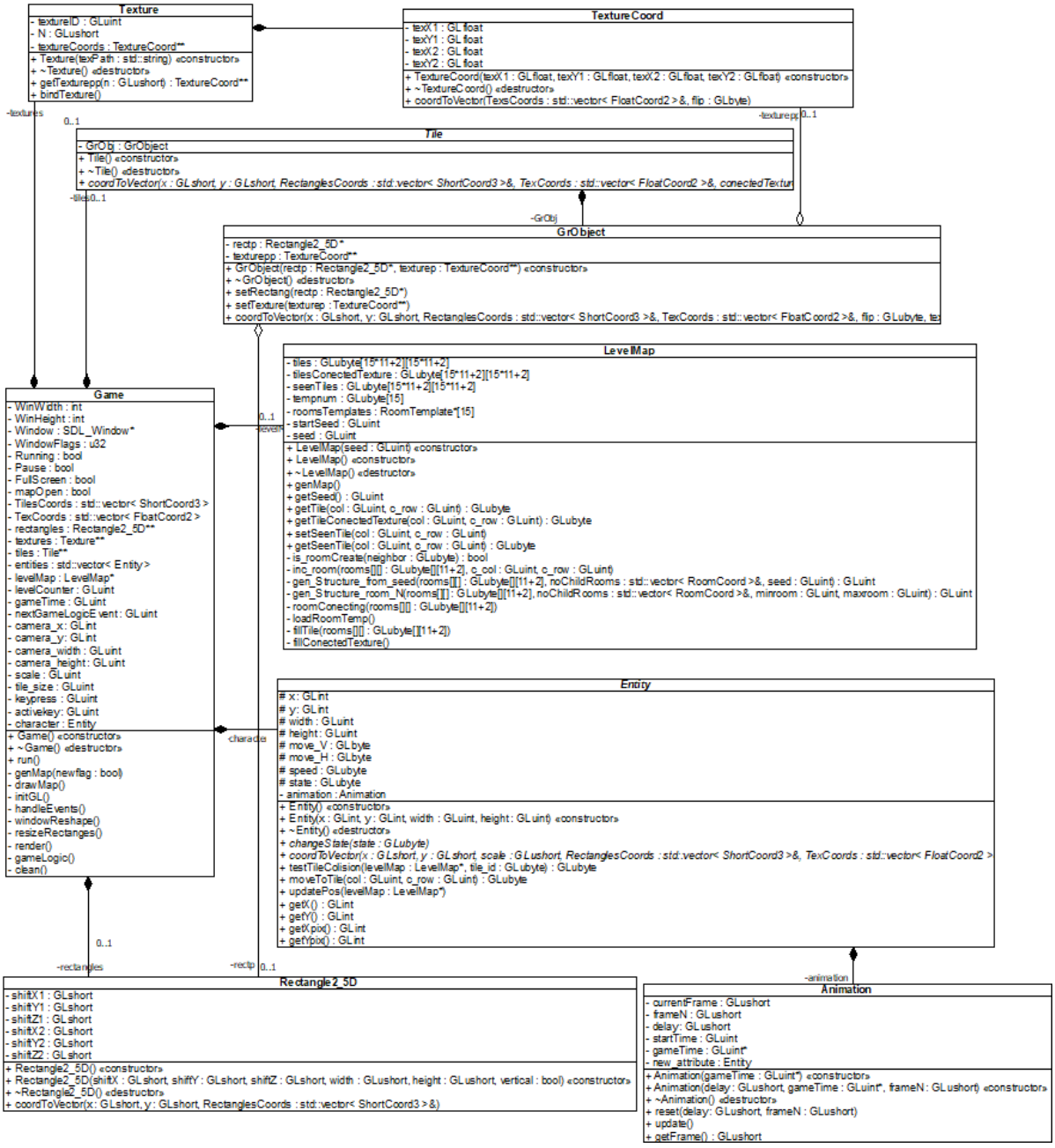


Рис.3.1.Діаграма класів

Основним класом ігрового додатку є клас Game, що відповідає за всі основні функції додатку. До складу даного класу входять наступні методи і атрибути:

- Game() – конструктор головного класу, що відповідає за ініціалізацію змінних відповідаючи за розмір вікна та стан додатку при запуску.

- void run() – метод, що відповідає за створення вікна та запуск основного циклу додатку.
- void genMap(bool newflag) – метод, що приймає як параметр значення newflag, що вказує необхідність створення нового набору рівнів з випадковим зерном.
- void handleEvents() – метод, що відповідає за опрацювання подій викликаних натисканням клавіш і змінами у стані вікна додатку.
- void initGL() – метод, що відповідає за ініціалізацію OpenGL, завантаження текстур у пам'ять відеоприскорювача і створення графічних об’єктів ігрового поля.
- void windowReshape() – метод, що опрацьовує подію зміни розміру вікна додатку. Для кожного нового розміру вікна обраховується нові розміри графічних об’єктів і нові значення для координатних осей.
- void resizeRectanges() – метод, що відповідає за перерахунок розмірів графічних об’єктів представлених класом Rectangle2_5D.
- void render() – метод, що відповідає за від рисовку зображення на екранні. Попереднє формування масивів, що містять в собі значення для виконання виведення на екран об’єктів, зменшує кількість запитів до відеоприскорювача.
- void drawMap() – метод відповідає за виведення на екран карти рівню, що поставо оновлюється по мірі його дослідження.
- void gameLogic() – метод завданням якого є перевірка взаємодії об’єктів, і зміни їх станів в залежності від ігрової логіки.
- void clean() – метод, що відповідає за очищення пам’яті після завершення виконання основного циклі додатку.
- int WinWidth – атрибут класу, що задає початкову ширину вікна додатку.

- int WinHeight – атрибут класу, що задає початкову висоту вікна додатку.
- SDL_Window *Window – атрибут класу, що зберігає в собі посилання на створене вікно.
- u32 WindowFlags – атрибут класу, що зберігає в собі прапори вікна. Ці данні використовуються у процесі зміни розміру вікна і переходу в повно екранний режим.
- bool Running – атрибут класу, що визначає чи додаток виконується в даний момент і є умовою до продовження виконання основного циклу.
- bool Pause – атрибут класу, що визначає чи додаток знаходиться в стані призупинення обрахунку основної ігрової логіки.
- bool FullScreen – атрибут класу, що відповідає за перехід вікна в повноекранний режим.
- bool mapOpen – атрибут класу, що визначає чи у процесі виводу на екран ігрового поля буде виведено карту рівня.
- std::vector<ShortCoord3> TilesCoords – атрибут класу, що зберігає координати для від рисовки полігонів з допомогою OpenGL.
- std::vector<FloatCoord2> TexCoords – атрибут класу, що містить текстурні координати для від рисовки полігонів з допомогою OpenGL.
- Rectangle2_5D** rectangles – атрибут класу, що містить посилання на масив екземплярів класу Rectangle2_5D. З їх допомогою реалізовано компактне зберігання координат полігонів. У випадку зміни масштабу при зміні розміру віна значення координат полігонів перераховуються.
- Texture** textures – атрибут класу, що містить посилання на масив екземплярів класу Texture.

- Tile** tiles – атрибут класу, що містить посилання на масив екземплярів класу Tile. Оскільки ігровий рівень складається з обмеженого набору об’єктів оточення в даному масиві зберігається по одному екземпляру кожного виду.
- Character* character – атрибут класу, що містить посилання на екземпляр класу Character. Даний об’єкт відповідає за представлення ігрового персонажа на екранні і взаємодію його з світом гри.
- LevelMap* levelMap – атрибут класу, що містить посилання на екземпляр класу LevelMap. Даний об’єкт зберігає в собі повну карту рівня і виконує генерацію нових рівнів.
- GLuint levelCounter – атрибут класу, що підраховує кількість пройдених гравцем рівнів.
- GLuint gameTime – атрибут класу, що зберігає в собі значення ігрового часу.
- GLint camera_x – атрибут класу містить координату x камери на ігровому полі.
- GLint camera_y – атрибут класу містить координату y камери на ігровому полі.
- GLuint camera_width – атрибут класу містить значення ширини камери.
- GLuint camera_height – атрибут класу містить значення висоти камери.
- GLuint scale – атрибут класу відповідає за масштабування об’єктів ігрового поля при зміні розміру вікна.
- GLuint tile_size – атрибут класу містить масштабоване значення розміру плитки на екрані в пікселях.
- GLuint keypress – атрибут класу зберігає стан клавіш, що відповідають за керування персонажем.

- GLuint activekey – атрибут класу визначає останню нажату клавішу в наборах клавіш, що не можуть обробитися одночасно.

3.1.1 Генерація рівнів

LevelMap є класом, що відповідає за генерацію рівня і зберігання всіх його додаткових параметрів. До складу класу входять такі методи і атрибути як:

- LevelMap(GLuint seed) – конструктор класу з параметром seed, що вказує зерно для генерації набору рівнів.
- LevelMap() – конструктор класу, що виконує аналогічні LevelMap(GLuint seed) функції, але як зерно використовує значення часу в момент виклику.
- ~LevelMap() – деструктор класу, що очищує пам'ять виділену для зберігання шаблонів кімнат.
- void genMap() – метод, що об’єднує всі кроки генерації рівня, і виконує генерацію наступного рівня в послідовності.
- void setSeenTile(GLuint col, GLuint c_row) – метод, що дозволяє відображення плитки на карті.
- GLuint getSeed() – метод, що повертає значення зерна для набору рівнів.
- GLubyte getTile(GLuint col, GLuint c_row) – метод, що повертає тип плитки.
- GLubyte getTileConectedTexture(GLuint col, GLuint c_row) – метод, що повертає тип текстури плитки.
- GLubyte getSeenTile(GLuint col, GLuint c_row) – метод, що повертає тип значення дозволу на відображення плитки на карті.
- bool is_roomCreate(GLubyte neighbor) – метод визначає чи буде розміщена кімната в комірці, що має кількість сусідніх зайнятих плиток рівну neighbor.

- void inc_room(GLubyte rooms[][R_COLUMN + 2], GLuint c_col, GLuint c_row) – метод, що виконує збільшення значення кількості сусідніх кімнат для сусідніх плиток.
- GLuint gen_Structure_from_seed(GLubyte rooms[][R_COLUMN + 2], std::vector<RoomCoord> &noChildRooms, GLuint seedGLuint gen_Structure_room_N(GLubyte rooms[][R_COLUMN + 2], std::vector<RoomCoord> &noChildRooms, GLuint minroom, GLuint maxroom); void roomConecting(GLubyte rooms[][R_COLUMN + 2]) – метод виконує розміщення кімнат в масиві rooms з відкиданням варіантів генерації, що не задовольняють умову minroom<кількість кімнат<maxroom.
- void loadRoomTemp() – метод, що виконує завантаження шаблонів для кімнат з файлу.
- void fillTile(GLubyte rooms[][R_COLUMN + 2]) – метод, що використовує шаблони кімнат для заповнення двовимірного масиву плиток значення ми у відповідності до значень типу кімнат, що зберігаються у двовимірному масиві rooms.
- void fillConectedTexture() – метод, що розраховує типи текстур для плиток на ігровому полі.
- GLubyte tiles[T_HEIGHT + 2][T_WIDTH + 2] – двовимірний масив, що мітить тип кожної плитки поля.
- GLubyte tilesConectedTexture[T_HEIGHT + 2][T_WIDTH + 2] – двовимірний масив, що містить номери текстур для кожної плитки поля.
- GLubyte seenTiles[T_HEIGHT + 2][T_WIDTH + 2] – двовимірний масив, що містить дозволи на відображення плиток на карті.
- GLubyte tempnum[15] – масив, що вказує кількість шаблонів для кожної різновидності кімнат.

- RoomTemplate* roomsTemplates[15] – масив, що містить посилання на масиви шаблонів кімнат.
- GLuint startSeed – атрибут, що зберігає значення зерна для даної послідовності рівнів.
- GLuint seed – атрибут, що зберігає значення зерна для останнього згенерованого рівня.

У процесі взаємодії з картою рівня і при її генерації використовуються такі структури як:

- SubMapCoord – структура даних, що використовується для розрахунку області карти, що перетинається з прямокутником заданої ширини і висоти.
- RoomCoord – структура даних, що об’єднує в собі номер стовпця і рядка у матриці.
- RoomTemplate – структура даних в якій зберігається шаблон кімнати.

3.1.2 Менеджер текстур

Texture є класом додатку, що відповідає за завантаження, зберігання і використання текстурного атласу.

- Texture(std::string texPath) – конструктор класу, що завантажує в пам'ять відеоприскорювача текстуру і виконує завантаження відповідних її текстурних координат.
- ~Texture() – деструктор класу, що виконує звільнення пам’яті відеоприскорювача і видалення з оперативної памяті текстурних координат.
- void bindTexture() – метод виконує завантаження текстури для її використання підчас виведення на екран об’єктів, що її використовують.

- TextureCoord** getTexturepp(GLushort n) – повертає посилання на текстурні координати, що знаходяться в n комірці масиву textureCoords.
- GLuint textureID – атрибут, що зберігає ындивыдуальний номер текстури в пам’яті відеоприскорювача.
- GLushort N – атрибут, що зберігає кількість текстурних координат.
- TextureCoord** textureCoords – посилання на масив екземплярів класу TextureCoord, що використовується для компактного зберігання текстурних координат.

Для зменшення затрат пам’яті на зберігання текстурних координат використовується клас TextureCoord. Оскільки всі під текстури в текстурному атласі мають прямокутну форму, то для їх зберігання достатньо двох наборів точок. Даний клас містить наступні методи і атрибути:

- TextureCoord(GLfloat texX1, GLfloat texY1, GLfloat texX2, GLfloat texY2) – конструктор класу, що приймає як параметри два набори координат, що визначають нижній лівий і верхній правий кут прямокутної підтекстури.
- void coordToVector(std::vector<FloatCoord2> &TexsCoords, GLbyte flip) – метод, що записує текстурні координати в масив для, що мітить відповідні до координат графічних об’єктів текстурні координати.
- GLfloat texX1 – атрибут містить значення координати x для нижнього лівого кута прямокутної підтекстури.
- GLfloat texY1 – атрибут містить значення координати y для нижнього лівого кута прямокутної підтекстури.
- GLfloat texX2 – атрибут містить значення координати x для верхнього правого кута прямокутної підтекстури.
- GLfloat texY2 – атрибут містить значення координати y для верхнього правого кута прямокутної підтекстури.

3.1.3 Графічні об’єкти

Для зображення на екрані використовуються прямокутні полігони, що представлені класом Rectangle2_5D. Оскільки кожний з полігонів є прямокутником вирівняним по осям при проекції в будь-яку площину, то для зберігання такого полігона в пам’яті достатньо двох наборів координат. Даний клас має такі методи і атрибути як:

- Rectangle2_5D() – пустий конструктор класу, що використовується для резервування пам’яті потрібного розміру, без виконання розрахунків, що присутні в основному конструкторі.
- Rectangle2_5D(GLshort shiftX, GLshort shiftY, GLshort shiftZ, GLushort width, GLushort height, bool vertical) – конструктор класу, що в залежності від значення прапора vertical може розраховувати значення полігон, що лежить в площині паралельній площині xOy чи xOy*.
- void coordToVector(GLshort x, GLshort y, std::vector<ShortCoord3> &RectanglesCoords) – метод, що записує значення координат полігону в масив координат полігонів для виведення на екран.
- GLshort shiftX1 – атрибут містить значення координати x для нижнього лівого кута прямокутної підтекстури.
- GLshort shiftY1 – атрибут містить значення координати y для нижнього лівого кута прямокутної підтекстури.
- GLshort shiftZ1 – атрибут містить значення координати z для нижнього лівого кута прямокутної підтекстури.
- GLshort shiftX2 – атрибут містить значення координати x для верхнього правого кута прямокутної підтекстури.
- GLshort shiftY2 – атрибут містить значення координати y для верхнього правого кута прямокутної підтекстури.
- GLshort shiftZ2 – атрибут містить значення координати z для верхнього правого кута прямокутної підтекстури.

GrObject є класом, що використовує класи Rectangle2_5D і TextureCoord для формування техтурованого полігону. Даний клас містить наступні методи і атрибути:

- GrObject(Rectangle2_5D* rectp, TextureCoord** texturep) – конструктор класу, що присвоює відповідним атрибутам посилання на екземпляр класу Rectangle2_5D і посилання на масив екземплярів класу TextureCoord. Завдяки посиланню на масив екземплярів класу TextureCoord можливе використання різних текстур для одного і того графічного об’єкта.
- void setRectang(Rectangle2_5D* rectp) – метод, що присвоює нове посилання на екземпляр Rectangle2_5D.
- void setTexture(TextureCoord** texturep) – метод, що присвоює нове посилання на масив екземплярів класу TextureCoord.
- void coordToVector(GLshort x, GLshort y, std::vector<ShortCoord3> &RectanglesCoords, std::vector<FloatCoord2> &TexCoords, GLubyte flip, GLubyte texShift) – метод, що записує значення координат полігону і відповідні їм значення текстурних координат в масиви для виведення на екран.
- Rectangle2_5D *rectp – атрибут, що містить посилання на екземпляр класу Rectangle2_5D.
- TextureCoord **texturepp – атрибут, що містить посилання на масив екземплярів класу TextureCoord..

3.1.4 Об’єкти ігрового поля

Статичні об’єкти ігрового поля представлені в ігровому додатку нащадками абстрактного класу Tile. Даний клас має методи абстрактний метод. virtual void coordToVector(GLshort x, GLshort y, std::vector<ShortCoord3> &RectanglesCoords, std::vector<FloatCoord2> &TexCoords, GLubyte conectedTexture)=0. Даний метод приймає як параметри координати розміщення об’єкту, посилання на масиви для об’єктів,що будуть

виведені на екран, і тип текстури, що буде використовуватись для відображення об’єкта.

Для представлення об’єктів ігрового поля, що динамічно змінюють свій зовнішній вигляд в залежності від часу використовується клас Animation, що в залежності від внутрішньо ігрового часу повертає відповідне значення зсуву в масиві текстурних координат. Даний клас містить наступні методи і атрибути:

- Animation(GLuint* gameTime) – конструктор класу, що зберігає посилання на змінну, що містить значення внутрішньо ігрового часу у відповідний атрибут класу.
- Animation(GLushort delay, GLuint* gameTime, GLushort frameN) – конструктор виконує дії аналогічні конструктору Animation(GLuint* gameTime) і встановлює значення, що відповідають за затримку зміни кадру і їх кількість.
- void reset(GLushort delay, GLushort frameN) – метод, що оновлює значення затримки зміни кадру і їх кількості.
- void update() – метод перераховує значення зсуву в масиві текстурних координат.
- GLushort getFrame() – метод, що повертає зсув в масиві текстурних координат.
- GLushort currentFrame – атрибут, що зберігає значення зсуву в масиві текстурних координат.
- GLushort frameN – атрибут, що зберігає кількість кадрів анімації.
- GLushort delay – атрибут, що зберігає затримку між кадрами анімації.
- GLuint startTime – атрибут, що зберігає час початку відтворення анімації.

- GLuint* gameTime – атрибут, що зберігає посилання на змінну, що містить значення внутрішньо ігрового часу.

Для представлення на ігровому полі об’єктів, що можуть змінювати свій стан і переміщуватись по ньому і взаємодіяти з іншими об’єктами використовуються нащадки класу Entity. Даний клас містить наступні методи і атрибути:

- Entity(GLint x, GLint y, GLuint width, GLuint height) – конструктор класу, що приймає як параметри початкову позицію об’єкта і розміри прямокутного об’єкта колізії, що йому відповідають.
- virtual void changeState(GLubyte state) = 0 – абстрактний метод, що відповідає за зміну стану об’єкта.
- virtual void coordToVector(GLshort x, GLshort y, GLushort scale, std::vector<ShortCoord3> &RectanglesCoords, std::vector<FloatCoord2> &TexCoords) = 0 – абстрактний метод, що приймає як параметри координати розміщення об’єкту на екрані, посилання на масиви для об’єктів, що будуть виведені на екран
- GLubyte testTileColision(LevelMap* levelMap, GLubyte tile_id) – метод, що перевіряє факт перетину з плитками карти певного типу.
- GLubyte moveToTile(GLuint col, GLuint c_row) – метод, що встановлює координати об’єкта такими, що відповідають координатам плитки певного рядка і стовпця карти.
- void updatePos(LevelMap* levelMap) – метод, що виконує переміщення об’єкта по ігровому полю з врахуванням претинів об’єкта з стінами.
- GLint getX() – метод, що повертає повне значенян координати x для об’єкта.
- GLint getY() – метод, що повертає повне значенян координати y для об’єкта.

- GLint getXpix() – метод, що повертає обрізане по кількості біт на піксель значенян координати x для об’єкта.
- GLint getYpix() – метод, що повертає обрізане по кількості біт на піксель значенян координати y для об’єкта.
-
- GLint x – атрибут, що мітить значення координати x об’єкта.
- GLint y – атрибут, що мітить значення координати y об’єкта.
- GLuint width – атрибут, що містить значення ширини об’єкта колізії.
- GLuint height – атрибут, що містить значення висоти об’єкта колізії.
- GLbyte move_V – атрибут, що вказує на тип переміщення вздовж осі Oy.
- GLbyte move_H – атрибут, що вказує на тип переміщення вздовж осі Ox.
- GLubyte speed – атрибут, що визначає швидкість переміщення об’єкта по ігровому полю.
- GLubyte state – атрибут, що визначає поточний стан об’єкта .

3.2 Тестування генератора рівнів

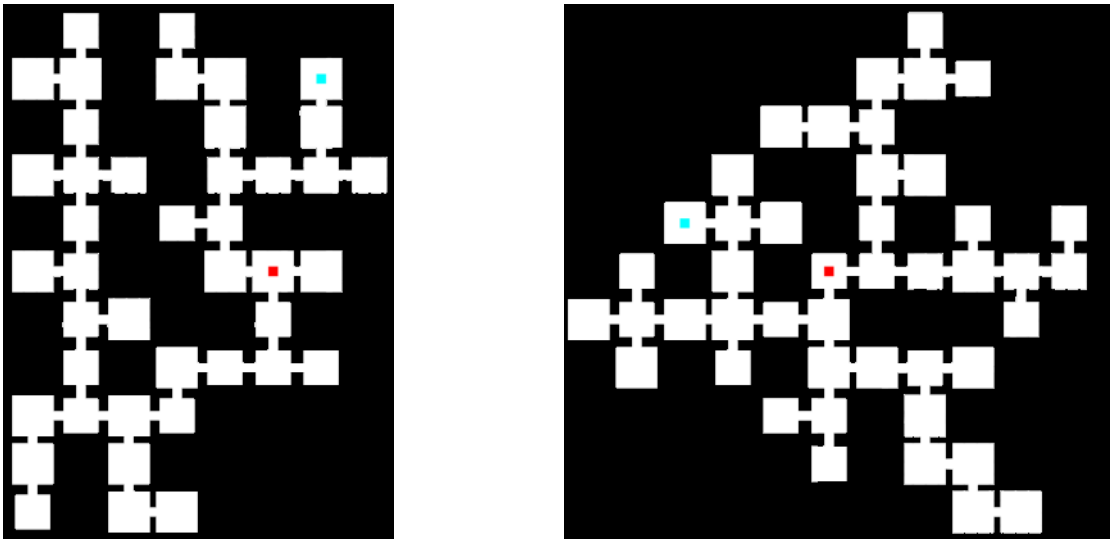
Для тестування роботи генератора рівнів було вирішено розглянути приклади генерації з однаковим значення зерна, що дорівнює “3”, і трьома різними варіантами коефіцієнтів, що відповідають за імовірність розміщення кімнати в комірці з певною кількістю зайнятих сусідніх комірок.

Для першого випробування було обрано значення коефіцієнтів, що наведені в таблиці 3.1.

Таблиці 3.1. Значення коефіцієнтів для першого випробування

Кількість сусідніх зайнятих комірок	Шанс розміщення кімнати в комірці
1	1
2	0.75
3	0.01
4	0.005

Для обраних значень коефіцієнтів було згенеровано два перші рівнів з послідовності, що зображені на рис 3.1.



а) б)

Рис.3.2.Приклад генерації з першим набором коефіцієнтів а)Перший рівень з набору рівнів; б) Другий рівень з набору рівнів

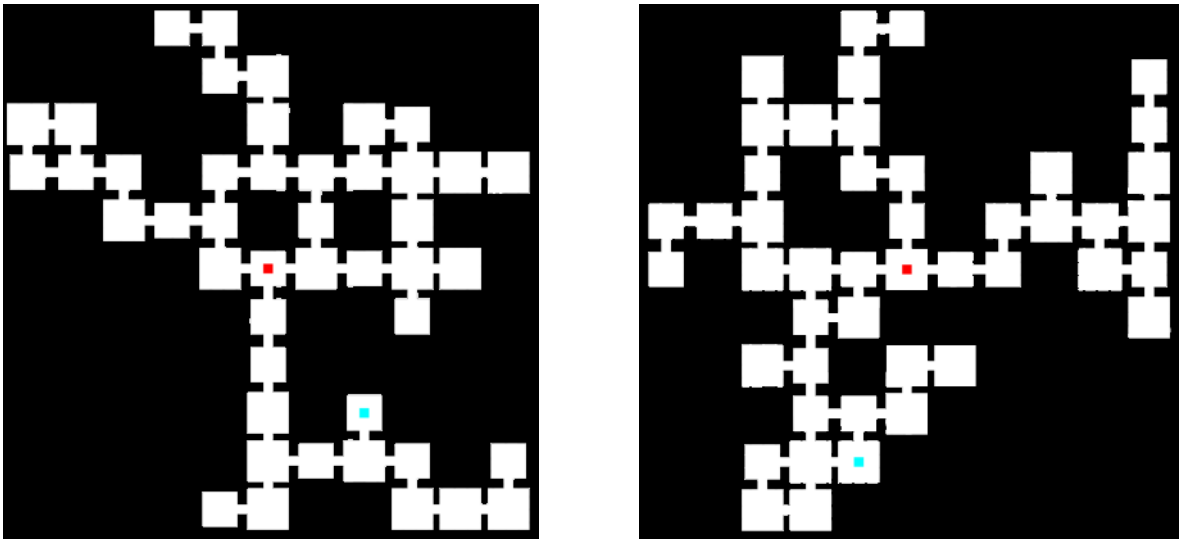
При розгляді згенерованих рівнів для першого випробування було зроблено висновок, що незважаючи на стовідсоткову імовірність генерації кімнати в комірках, що мають одну зайняту сусідню комірку, завдяки випадковому вибору порядку утворення нащадків при кожному запуску створюється унікальний рівень.

Для другого випробування було обрано значення коефіцієнтів, що наведені в таблиці 3.2.

Таблиці 3.2. Значення коефіцієнтів для другого випробування

Кількість сусідніх зайнятих комірок	Шанс розміщення кімнати в комірці
1	0.6
2	0.4
3	0.1
4	0.005

Для обраних значень коефіцієнтів було згенеровано два перші рівнів з послідовності, що зображені на рис 3.2.



а) б)

Рис.3.3.Приклад генерації з другим набором коефіцієнтів а)Перший рівень з набору рівнів; б) Другий рівень з набору рівнів

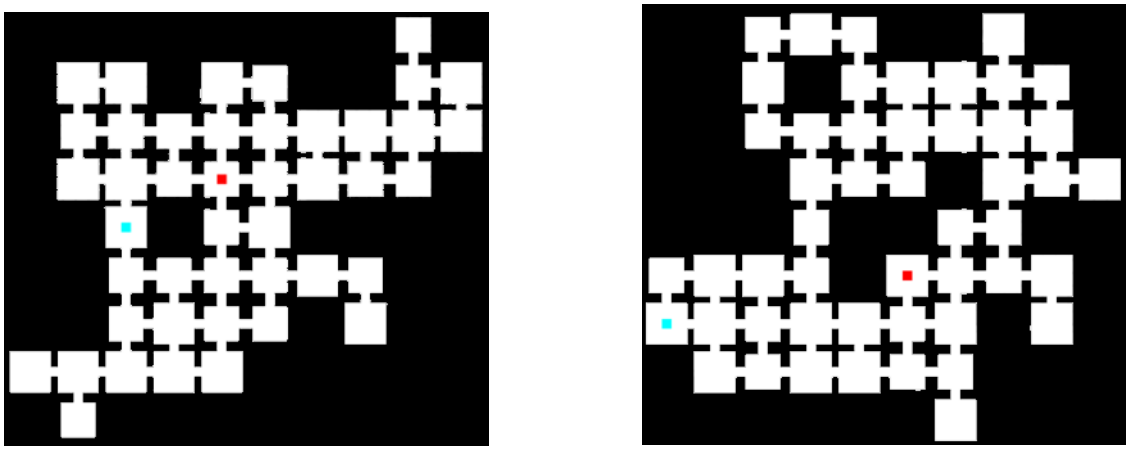
При розгляді згенерованих рівнів для другого випробування було помічено можливість генерації невеликих нагромадження кімнат, що робить даний набір коефіцієнтів недоцільним для даної версії алгоритму. В той же час при реалізації восьми бітного визначення типу кімнат можливе створення рівнів з кімнатами різноманітного розміру і форми. Іншим можливим рішенням є врахування кутових сусідів при розміщенні кімнат.

Для третього випробування було обрано значення коефіцієнтів, що наведені в таблиці 3.3.

Таблиці 3.3. Значення коефіцієнтів для третього випробування

Кількість сусідніх зайнятих комірок	Шанс розміщення кімнати в комірці
1	0.6
2	0.4
3	0.3
4	0.005

Для обраних значень коефіцієнтів було згенеровано два перші рівнів з послідовності, що зображені на рис 3.3.



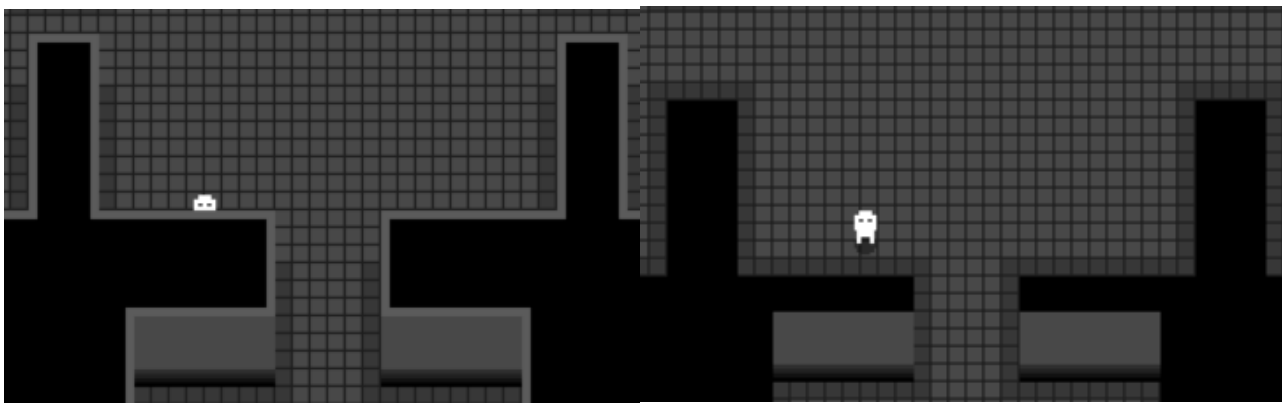
а) б)

Рис.3.4. Приклад генерації з третім набором коефіцієнтів а)Перший рівень з набору рівнів; б) Другий рівень з набору рівнів

При розгляді згенерованих рівнів для третього випробування було визначено недоцільність використання заданої величини імовірності генерації кімнати з трьома сусідами і необхідність встановлення її такою, що майже не впливає на генерацію. В той же час даний експеримент показав недоцільність генерації кімнат з чотирма сусідніми зайнятими комірками.

3.3 Тестування графічної складової

Для розгляду графічної складової додатку було розглянуто дві аналогічні карти з відображенням стін і їх відсутністю, що показано на рис.3.5.



а) б)

Рис.3.5 Приклад ігрового поля а) з перекриттям персонажа стіною; б) з відключеним відображенням верхньої частини стін

Представлені рисунки показують успішність реалізації графічної складової додатку. На рис.3.5а представлено правильність прикриття одного об’єкта ігрового поля іншим, а на рис.3.5б правильність з’єднання текстур плиток підлоги.

Для визначення затрат ресурсів, додаток було протестовано комютері зпроцесором Intel Core i5-7200U 2.50GHz і відеокартою Intel HD Graphics 620. При тестуванні ігровий додаток віконному режимі з розширенням вікна 600x400 у моментах пікового навантаження використовував 15% ресурсу процесора, 12Мб оперативної памяті і 30% ресурсу графічного прискорювача. При тестуванні ж у повно віконному режимі з розширенням екрану 1920x1080 було виявлено відсутність збільшення затрат ресурсів процесора і оперативної пам’яті, але використання графічного прискорювача виросло до 50%.Підчас виконання тестування частота кадрів додатку стабільно була рівною 250 кадрам на секунду, що рівно її обмеженню для додатку.

.

Висновок до розділу 3

В першій частині даного розділу було описано структуру основних класів створеного додатку і приведено спрощену діаграму класів. Окрім цього було приведено способи зменшення затрат пам’яті, шляхом повторного використання об’єктів .

В другій частині розділу даного розділу було проведено аналіз результатів генерації рівнів з різними параметрами створеного генератора і описано помічені особливості генерації. У процесі розгляду було виявлено, що для використання коефіцієнтів, що передбачають генерацію кімнат з в комірках з двома зайнятими сусідніми комірками є доцільним реалізація восьми бітного механізму з’єднання кімнат. Для цього необхідно буде створити 47 шаблонів для кімнат, що збільшить кількість затрат на створення рукотворних об’єктів втричі, але зможе створити значно більшу варіативність карти.

В третій частині даного розділу було описано результати роботи графічної складової додатку. У процесі розгляду були протестовані механізми перекриття об’єктів і з’єднання текстур плиток, з яких складається ігрове поле. Не було виявлено жодних помилок у логіці представлення об’єктів на екрані, що доводить правильність висунутих рішень.

.

ЗАГАЛЬНІ ВИСНОВКИ

У результаті виконання дипломної роботи було розроблено ігровий додаток з програмною генерацією рівнів. Розроблений додаток виконує всі поставлені перед ним завдання. У додатку виконується генерація унікальних рівнів при кожному наступному запуску додатку і при цьому зберігається можливість генерації точної копії згенерованого рівню з використанням аналогічного значення зерна.

У процесі розробки додатку було вирішено використовувати парадигму об’єктно орієнтованого програмування, що дозволило згрупувати основні функції додатку в окремих класах і ти самим зробити більш прозорою структуру додату.

Для вирішення завдання генерації рівнів було розроблено алгоритм генерації від центра сітки, що може налаштовуватись з допомогою коефіцієнтів імовірності створення кімнати з n кількістю сусідів. Отриманий алгоритм добре виконує свої основні функції, але має безліч можливостей для вдосконалення. Основною з них є використання восьми бітного формату типу кімнати, що аналогічно до з’єднання текстур плиток дозволить реалізувати більш якісне з’єднання кімнат і можливість побудови кімнат різноманітного розміру. Недоліком цього кроку є необхідність створення сорока семи типів шаблонів заміть п’ятнадцяти, що збільшить затрати часу. Іще одним можливим вдосконаленням алгоритму є зміна способу врахування сусідніх зайнятих комірок матриці кімнат, що може покращити якість розміщення кімнат, що мають більше одного сусіда, а також врахування не тільки кімнат зі спільними гранями, а й кімнат зі спільними вершинами. Такий крок дозволить краще формувати кімнати великого розміру, але потребує врахування нерівноцінності сусідніх кімнат.

Для додатку було розроблену методику графічного представлення двовимірних об’єктів у трьохвимірному просторі, що дозволяє реалізувати

перекриття об’єктів ігрового поля і дає потенціальну можливість використання розрахунку різноманітних з допомогою можливостей відеоприскорювача.

Додаток було протестовано на швидкодію і затрату ресурсів. В процесі тестування значення затрат ресурсів не перевищувало очікуванні і піддавалось логічному осмисленню. При зміні розширення вікна не було виявлено зростання затрат ресурсів центрального процесора, що вказує на коректність використання ресурсів відео прискорювача.

Для розробки додатку було використано бібліотеку SDL і програмний інтерфейс OpenGL, що дозволяє скомпілювати отриманий додаток для виконання на платформах Microsoft Windows, Linux і Android., а у якості мови програмування використовувалась мова C++, що надало можливість точної роботи з пам’яттю.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Spelunky Generator Lessons [Електронний ресурс] – Tiny Subversions, 2013 – Режим доступу: http://tinysubversions.com/spelunkyGen/
2. Shaun M. SDL Game Development / Mitchell Shaun., 2013/ ISBN 978-1849696821, 258 с.
3. Adams T. Procedural Generation in Game Design / Tarn Adams, Tanya X. Short, 2017/ ISBN 978-1498799195, 336 с.
4. Shaker N. Procedural Content Generation in Games (Computational Synthesis and Creative Systems) / Noor Shaker, Julian Togelius, Mark J. Nelson., 2016/ ISBN 978-3319427140, 253 с.
5. Kessenich J. The OpenGL® Programming Guide 9th Edition / John Kessenich, Graham Sellers, and Dave Shreiner / ISBN 978-0321773036 , 2013. – 984 с.
6. Dungeon generation using BSP trees [Електронний ресурс] – eskerda.com, 2013 – Режим доступу: https://eskerda.com/bsp-dungeon-generation/
7. Mapgen: Tunneling Algorithm [Електронний ресурс] – gridsagegames.com, 2014 – Режим доступу: https://www.gridsagegames.com/blog/2014/06/mapgen-tunneling-algorithm/
8. Generate Random Cave Levels Using Cellular Automata [Електронний ресурс] – tutsplus.com, 2013 – Режим доступу: https://gamedevelopment.tutsplus.com/tutorials/generate-random-cave-levels-using-cellular-automata--gamedev-9664
9. Procedural Dungeon Generation: A Drunkard's Walk in ClojureScript [Електронний ресурс] – jrheard's blog, 2016 – Режим доступу: https://blog.jrheard.com/procedural-dungeon-generation-drunkards-walk-in-clojurescript
10. Adventures in Bitmasking [Електронний ресурс] – angryfishstudios.com, 2011 – Режим доступу: http://www.angryfishstudios.com/2011/04/adventures-in-bitmasking/

11. Introductory Guide to AABB Tree Collision Detection [Електронний ресурс] – azurefromthetrenches.com, 2017– Режим доступу:https://www.azurefromthetrenches.com/introductory-guide-to-aabb-tree-collision-detection/

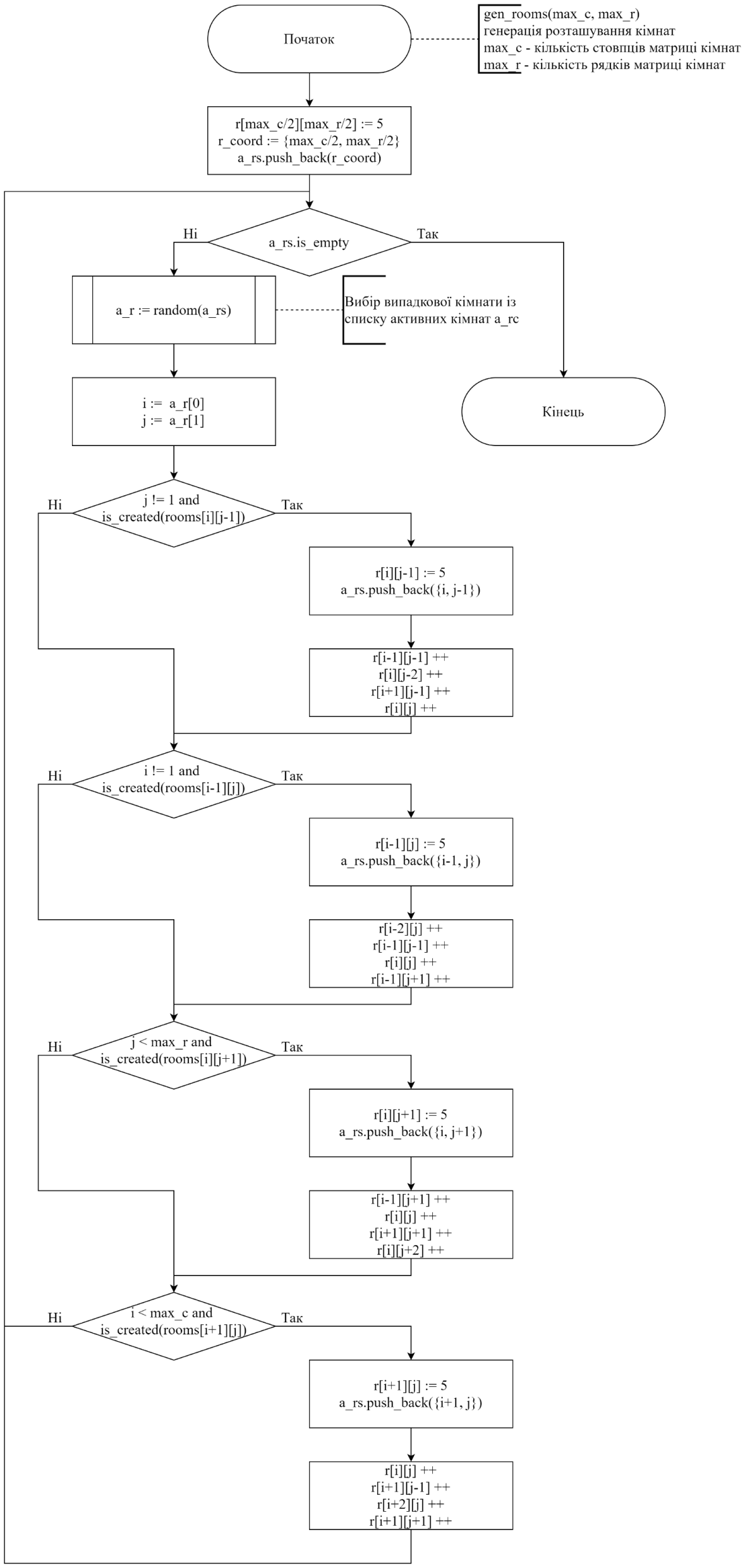
ДОДАТОК А

“Ігровий додаток з генерацією рівнів”

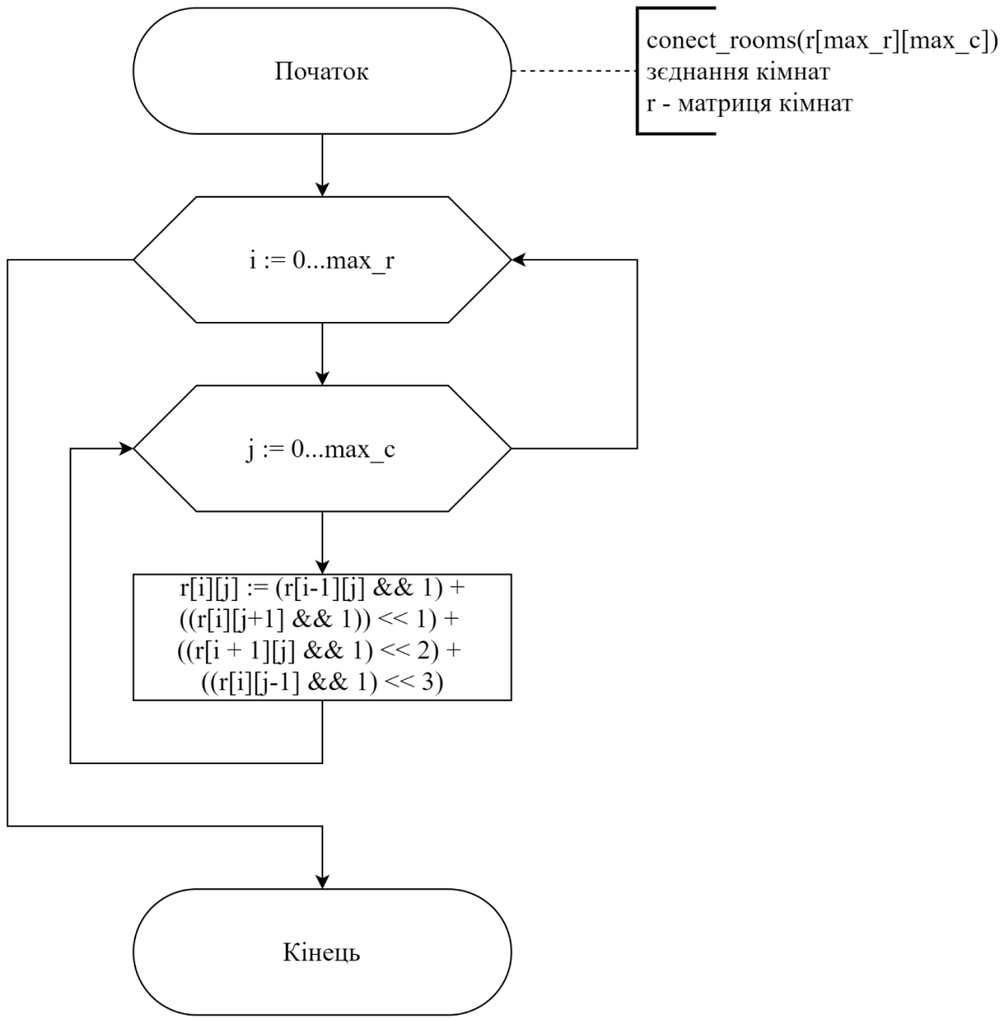
КОПІЇ ГРАФІЧНИХ МАТЕРІАЛІВ

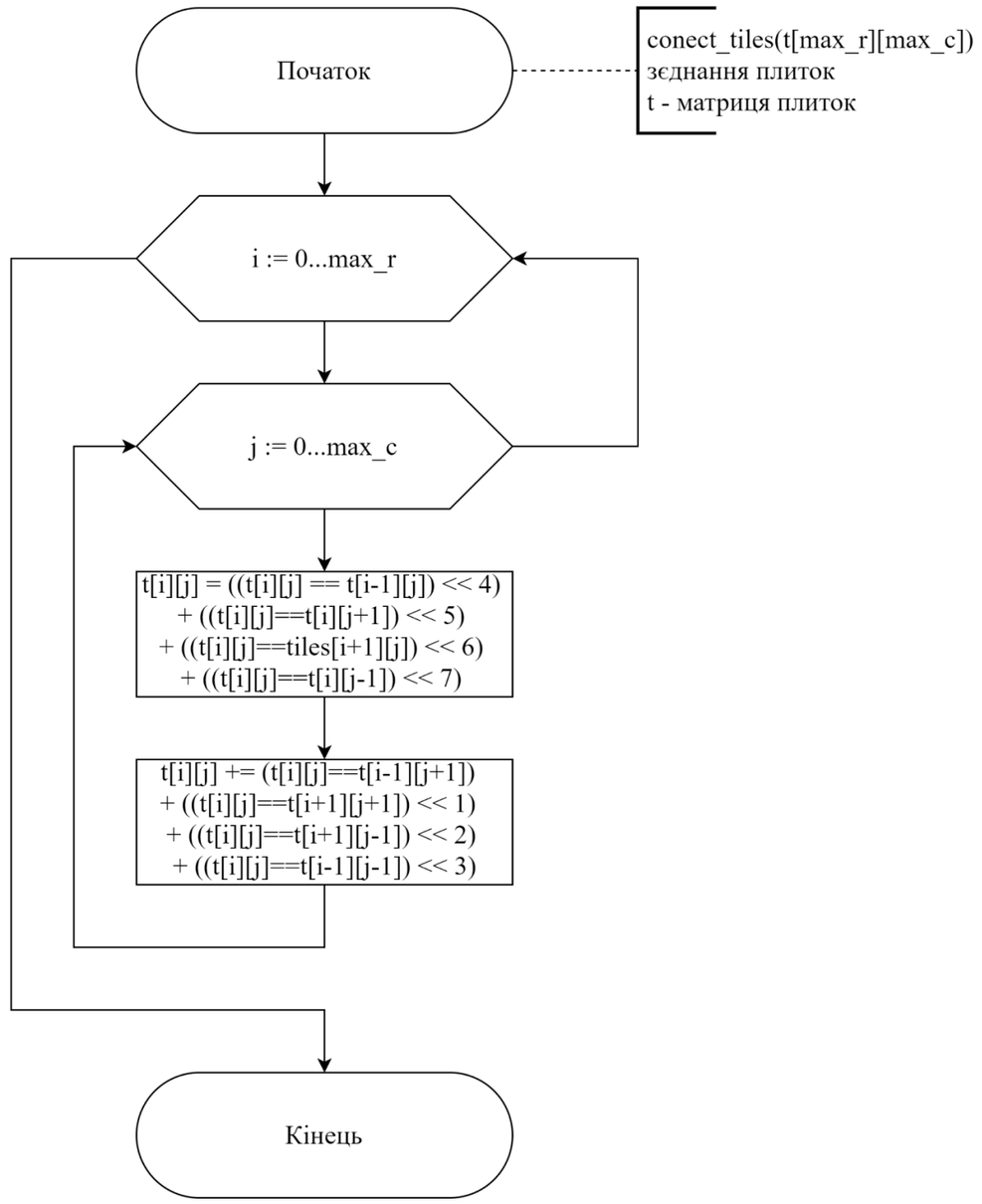
Аркушів 3

Київ 2020

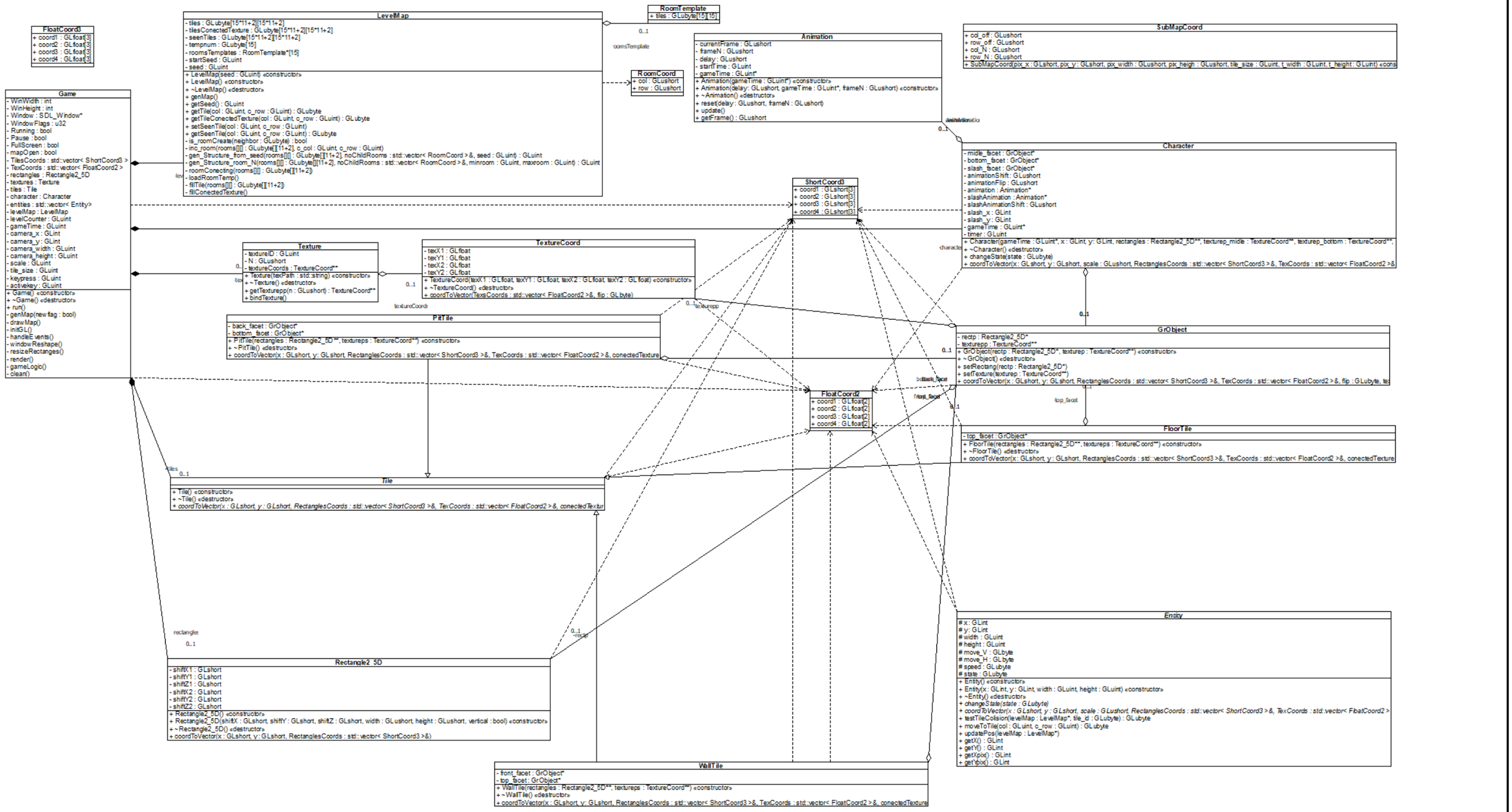


					ІА/ЛЦ.467800.004 Д1	
					Ігровий додаток з генерацією рівнів	Літера / Маса / Масштаб
Зм.	Арк.	№ докум	Підпис	Дата	Блок-схема алгоритму заповнення матриці кімнат	
Розробив		Сухобрус О.І				Аркуш 1 / Аркушів 1
Перевірив		Регіда П.Г.				
Т. контр.						
Н. контр.		Сімоненко В.П.			Дипломний проект	НТУУ "КПІ ім. Ігоря Сікорського", ФІОТ, ІО-62
Затверд.						





					ІА/ЛЦ.467800.005 Д2	
					Ігровий додаток з генерацією рівнів	Літера / Маса / Масштаб
Зм.	Арк.	№ докум	Підпис	Дата	Блок-схеми алгоритмів з'єднання сусідніх об'єктів матриці	
Розробив		Сухобрус О.І				Аркуш 1 / Аркушів 1
Перевірив		Регіда П.Г.			Дипломний проект	НТУУ "КПІ ім. Ігоря Сікорського", ФІОТ, ІО-62
Т. контр.						
Н. контр.		Сімоненко В.П.				
Затверд.						



					ІА/ЛЦ.46 7800.006 Д3	
Зм.	Арк.	№ докум	Підпис	Дата	Ігровий додаток з генерацією рівнів	Літера / Маса / Масштаб
Розробив		Сухобрус О.І			Діаграма класів	Аркуш 1 / Аркушів 1
Перевірив		Регіда П.Г.			Дипломний проект	НТУУ "КПІ ім. Ігоря Сікорського", ФІОТ, ІО-62
Т. контр.						
Н. контр.		Сімоненко В.П.				
Затверд.						

ДОДАТОК В

“Ігровий додаток з генерацією рівнів”

ЛІСТИНГ ПРОГРАМИ

Аркушів 20

Київ 2020

include\Game.h

```
#ifndef GAME_H
#define GAME_H
#include <SDL.h>
#include <SDL_opengl.h>
#include <GL/gl.h>
#include <assert.h>
#include <vector>
#include <ctime>
#include <iostream>
#include "Texture.h"
#include "LevelMap.h"
#include "FloorTile.h"
#include "WallTile.h"
#include "PitTile.h"
#include "Character.h"
#define MAX_ELM_HEIGHT 256
#define MINIMAP_SIZE 2
#define CAMERA_TILE_SIZE 10
#define FPS 60
#define GAME_LOGIC_DELAY 10
const GLuint FRAME_TIME = 1000/FPS;
typedef uint32_t u32;
class Game
{
public:
    Game();
    ~Game();
    void run();
private:
    int WinWidth;
    int WinHeight;
    SDL_Window *Window;
    u32 WindowFlags;
    bool Running;
    bool Pause;
    bool FullScreen;
    bool mapOpen;
    std::vector<ShortCoord3> TilesCoords;
    std::vector<FloatCoord2> TexCoords;
    Rectangle2_5D** rectangles;
    Texture** textures;
    Tile** tiles;
    Character* character;
    std::vector<Entity> entities;
    LevelMap* levelMap;
    GLuint levelCounter;
    GLuint gameTime;
    GLint camera_x;
    GLint camera_y;
```

```
    GLuint camera_width;
    GLuint camera_height;
    GLuint scale;
    GLuint tile_size;
    GLuint keypress;
    GLuint activekey;
    void genMap(bool newflag);
    void drawMap();
    void initGL();
    void handleEvents();
    void windowReshape();
    void resizeRectanges();
    void render();
    void gameLogic();
    void clean();

};
#endif // GAME_H
```

stc\Game.h

```
#include "Game.h"
Game::Game()
{
    this->WinWidth = 600;
    this->WinHeight = 400;
    this->levelCounter = 0;
    this->Pause = false;
    this->mapOpen = false;
    this->camera_x = 0;//1000;
    this->camera_y = 0;//2000;
    this->keypress = 0;
    this->activekey = 0;
}
Game::~Game()
{
    //dtor
}
void Game::handleEvents()
{
    SDL_Event Event;
    while (SDL_PollEvent(&Event))
    {
        if (Event.type == SDL_KEYDOWN)
        {
            switch (Event.key.keysym.sym)
            {
            case SDLK_w:
                this->activekey = this->activekey & ~3 | 1;
                this->keypress |= 1;
                break;
            case SDLK_s:
```

```
                this->activekey = this->activekey & ~3 | 2;
                this->keypress |= 2;
                break;
            case SDLK_d:
                this->activekey = this->activekey & ~12 | 4;
                this->keypress |= 4;
                break;
            case SDLK_a:
                this->activekey = this->activekey & ~12 | 8;
                this->keypress |= 8;
                break;
            case SDLK_UP:
                this->keypress &= 0xf;
                this->keypress |= 16;
                break;
            case SDLK_DOWN:
                this->keypress &= 0xf;
                this->keypress |= 32;
                break;
            case SDLK_RIGHT:
                this->keypress &= 0xf;
                this->keypress |= 64;
                break;
            case SDLK_LEFT:
                this->keypress &= 0xf;
                this->keypress |= 128;
                break;
            case SDLK_t:
                std::cout << "p: " << std::hex << this->keypress <<
std::endl;
                std::cout << "u: " << std::hex << this->activekey <<
std::endl;
                //this->character->testTileColision(this->levelMap,
0);
                this->character->updatePos(this->levelMap);
                break;
            case SDLK_r:
                this->genMap(true);
                this->levelCounter = 0;
                break;
            case SDLK_p:
                this->Pause = !this->Pause;
                break;
            case SDLK_m:
                this->mapOpen = !this->mapOpen;
                break;
            case 'f':
                this->FullScreen = !this->FullScreen;
                if (FullScreen)
                {
```

```
                    SDL_SetWindowFullscreen(this->Window, this-
>WindowFlags | SDL_WINDOW_FULLSCREEN_DESKTOP);
                }
                else
                {
                    SDL_SetWindowFullscreen(this->Window, this-
>WindowFlags);
                }
                break;
            case SDLK_ESCAPE:
                this->Running = false;
                break;
            default:
                break;
            }
        }
        else if (Event.type == SDL_KEYUP)
        {
            switch (Event.key.keysym.sym)
            {
            case SDLK_w:
                this->activekey = this->activekey & ~3 | 2;
                this->keypress &= ~1;
                break;
            case SDLK_s:
                this->activekey = this->activekey & ~3 | 1;
                this->keypress &= ~2;
                break;
            case SDLK_d:
                this->activekey = this->activekey & ~12 | 8;
                this->keypress &= ~4;
                break;
            case SDLK_a:
                this->activekey = this->activekey & ~12 | 4;
                this->keypress &= ~8;
                break;
            case SDLK_UP:
                this->keypress &= ~16;
                break;
            case SDLK_DOWN:
                this->keypress &= ~32;
                break;
            case SDLK_RIGHT:
                this->keypress &= ~64;
                break;
            case SDLK_LEFT:
                this->keypress &= ~128;
                break;
            default:
                break;
```

```
            }
        }
        else if (Event.type == SDL_WINDOWEVENT)
        {
            switch (Event.window.event)
            {
            case SDL_WINDOWEVENT_SIZE_CHANGED:
                this->windowReshape();
                this->resizeRectanges();
                if(this->Pause)
                    this->render();
                break;

            case SDL_WINDOWEVENT_FOCUS_LOST:
                this->Pause = true;
                break;

            default:
                break;
            }
        }
        else if (Event.type == SDL_QUIT)
        {
            this->Running = false;
        }
    }
}
void Game::initGL()
{
    //depth test
    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LEQUAL);
    //use alpha
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    //cull back faces CCW polygon
    glEnable(GL_CULL_FACE);
    glCullFace(GL_BACK);
    glFrontFace(GL_CCW);
    glEnableClientState(GL_VERTEX_ARRAY);
    glEnableClientState(GL_TEXTURE_COORD_ARRAY);
    glEnable(GL_TEXTURE_2D);
    //set window coordinate system and reset tiles
    windowReshape();
    //rectangle memory allocate
    #define rect_size 8
    this->rectangles = new Rectangle2_5D*[rect_size];
    for(int i = 0; i < rect_size; i++)
        this->rectangles[i] = new Rectangle2_5D();
    //set rectangle
```

```
    this->resizeRectanges();

    this->textures = new Texture*[2];
    this->textures[0] = new Texture("Textures\\Tiles");
    this->textures[1] = new Texture("Textures\\Entities");

    this->tiles = new Tile*[3];
    this->tiles[0] = new FloorTile( rectangles, this->textures[0]-
>getTexturepp(0)); //floor
    this->tiles[1] = new WallTile(  rectangles, this->textures[0]-
>getTexturepp(0)); //wall
    this->tiles[2] = new PitTile(   rectangles, this->textures[0]-
>getTexturepp(0)); //pit

    //this->texture = new Texture("Textures\\Character");
    //this->texture = new Texture("Textures\\Terrain");
}
void Game::resizeRectanges()
{
    new (this->rectangles[0]) Rectangle2_5D(0,      0,
0,                      this->tile_size,    this->tile_size,
false); //floor top
    new (this->rectangles[1]) Rectangle2_5D(0,      0,
0,                      this->tile_size,    this->tile_size*2,  true);
//wall front
    new (this->rectangles[2]) Rectangle2_5D(0,      0,
this->tile_size*2,      this->tile_size,    this->tile_size,
false); //wall top
    new (this->rectangles[3]) Rectangle2_5D(0,      this->tile_size,
-this->tile_size*2,     this->tile_size,    this->tile_size*2,  true);
//pit wall front
    new (this->rectangles[4]) Rectangle2_5D(0,      0,
-this->tile_size*2,     this->tile_size,    this->tile_size,
false); //pit bottom
    new (this->rectangles[5]) Rectangle2_5D(0,      this->tile_size/2,
0,      this->tile_size,    this->tile_size,    true); //character
    new (this->rectangles[6]) Rectangle2_5D(0,      0,
0,                      this->tile_size,    this->tile_size,
false); //shadow
    new (this->rectangles[7]) Rectangle2_5D(0,      0,
this->tile_size/2,      this->tile_size,    this->tile_size,
false); //slash
}
void Game::clean()
{
    delete this->levelMap;
    for(int i = 0; i < 3; i++)
        delete this->rectangles[i];
    delete[] this->rectangles;
    for(int i = 0; i < 2; i++)
```

```
        delete this->textures[i];
    delete[] this->textures;
    for(int i = 0; i < 2; i++)
        delete this->tiles[i];
    delete[] this->tiles;
    delete this->character;
}

void Game::windowReshape()
{
    SDL_GetWindowSize(Window, &this->WinWidth, &this->WinHeight);
    glViewport(0, 0, WinWidth, WinHeight);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    float aspect = (float)WinWidth / WinHeight;
    if (aspect >= 1.0)
    {
        if(WinHeight > CAMERA_TILE_SIZE*TILE_PIX_SIZE)
            this->scale = WinHeight /
(CAMERA_TILE_SIZE*TILE_PIX_SIZE);
        else
            this->scale = 1;
        this->camera_height = WinHeight;
        this->camera_width = this->camera_height * aspect;
    }
    else
    {
        if(WinWidth > CAMERA_TILE_SIZE*TILE_PIX_SIZE)
            this->scale = WinWidth / (CAMERA_TILE_SIZE*TILE_PIX_SIZE);
        else
            this->scale = 1;
        this->camera_width = WinWidth;
        this->camera_height = this->camera_width / aspect;
    }
    glOrtho(0, this->camera_width, 0,  this->camera_height, -
MAX_ELM_HEIGHT, MAX_ELM_HEIGHT);
    //glOrtho(0, WinWidth         , 0, WinHeight, -128, 0);
    this->tile_size = this->scale*TILE_PIX_SIZE;
}
void Game::render()
{
    glClearColor(0.f, 0.f, 0.f, 0.f);
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    this->camera_x = this->character->getXpix() - this->camera_width /
this->scale / 2;
    this->camera_y = this->character->getYpix() - this->camera_height
/ this->scale / 2;
    SubMapCoord camera(this->camera_x - TILE_PIX_SIZE*2 , this-
>camera_y - TILE_PIX_SIZE*2, this->camera_width / this->scale +
```

```
TILE_PIX_SIZE*2, this->camera_height / this->scale + TILE_PIX_SIZE*2,
TILE_PIX_SIZE, T_WIDTH, T_HEIGHT);
    //render map
    this->textures[0]->bindTexture();
    GLuint scale_camera_x = this->camera_x*this->scale;
    GLuint scale_camera_y = this->camera_y*this->scale;
    //-----------------------------------------
    for(int i = camera.row_off;  i < camera.row_N; i++)
    {
        for(int j =  camera.col_off; j < camera.col_N; j++)
        {
            GLbyte tileId = this->levelMap->getTile(j, i);
            this->levelMap->setSeenTile(j, i);
            if(tileId != 0)
            {
                this->tiles[tileId - 1]->coordToVector(j*this-
>tile_size - scale_camera_x, i*this->tile_size - scale_camera_y,
TilesCoords, TexCoords, this->levelMap->getTileConectedTexture(j, i));
            }
        }
    }
    //-----------------------------------------
    glVertexPointer(3, GL_SHORT, 0, &TilesCoords.front());
    glTexCoordPointer(2, GL_FLOAT, 0, &TexCoords.front());

    glDrawArrays(GL_QUADS, 0, TilesCoords.size()*4);
    TilesCoords.clear();
    TexCoords.clear();
    //render entity
    this->textures[1]->bindTexture();
    //-----------------------------------------
    this->character->coordToVector(-scale_camera_x, -scale_camera_y,
this->scale, TilesCoords, TexCoords);
    //-----------------------------------------
    glVertexPointer(3, GL_SHORT, 0, &TilesCoords.front());
    glTexCoordPointer(2, GL_FLOAT, 0, &TexCoords.front());
    glDrawArrays(GL_QUADS, 0, TilesCoords.size()*4);
    TilesCoords.clear();
    TexCoords.clear();
    if(this->mapOpen)
    {
        this->drawMap();
    }
    SDL_GL_SwapWindow(Window);
}

void Game::drawMap()
{
    std::vector<ShortCoord3> MapTilesCoords;
    std::vector<FloatCoord3> MapTilesColor;
```

```
    for(int i = 0;  i < T_WIDTH; i++)
    {
        for(int j =  0; j < T_HEIGHT; j++)
        {
            if(this->levelMap->getSeenTile(j, i) && this->levelMap-
>getTile(j,i) == 1)
            {
                MapTilesCoords.push_back({{j*MINIMAP_SIZE,
i*MINIMAP_SIZE, MAX_ELM_HEIGHT - 1},
{(j+1)*MINIMAP_SIZE, i*MINIMAP_SIZE, MAX_ELM_HEIGHT - 1},
{(j+1)*MINIMAP_SIZE, (i+1)*MINIMAP_SIZE, MAX_ELM_HEIGHT -
1},{j*MINIMAP_SIZE, (i+1)*MINIMAP_SIZE, MAX_ELM_HEIGHT - 1}});
                MapTilesColor.push_back({{1.0f, 1.0f, 1.0f},
                                         {1.0f, 1.0f, 1.0f},
                                         {1.0f, 1.0f, 1.0f},
                                         {1.0f, 1.0f, 1.0f}} );
            }else if(this->levelMap->getSeenTile(j, i) && this-
>levelMap->getTile(j,i) == 3)
            {
                MapTilesCoords.push_back({{j*MINIMAP_SIZE,
i*MINIMAP_SIZE, MAX_ELM_HEIGHT - 1},
                                          {(j+1)*MINIMAP_SIZE,
i*MINIMAP_SIZE, MAX_ELM_HEIGHT - 1},
                                          {(j+1)*MINIMAP_SIZE,
(i+1)*MINIMAP_SIZE, MAX_ELM_HEIGHT - 1},
                                          {j*MINIMAP_SIZE,
(i+1)*MINIMAP_SIZE, MAX_ELM_HEIGHT - 1}});
                MapTilesColor.push_back({{0.0f, 1.0f, 1.0f},
                                         {0.0f, 1.0f, 1.0f},
                                         {0.0f, 1.0f, 1.0f},
                                         {0.0f, 1.0f, 1.0f}} );
            }
        }
    }
    MapTilesCoords.push_back({{(this->character->getXpix() /
TILE_PIX_SIZE - 1)*MINIMAP_SIZE, (this->character->getYpix() /
TILE_PIX_SIZE - 1)*MINIMAP_SIZE, MAX_ELM_HEIGHT},
                              {(this->character->getXpix() /
TILE_PIX_SIZE + 2)*MINIMAP_SIZE, (this->character->getYpix() /
TILE_PIX_SIZE - 1)*MINIMAP_SIZE, MAX_ELM_HEIGHT},
                              {(this->character->getXpix() /
TILE_PIX_SIZE + 2)*MINIMAP_SIZE, (this->character->getYpix() /
TILE_PIX_SIZE + 2)*MINIMAP_SIZE, MAX_ELM_HEIGHT},
                              {(this->character->getXpix() /
TILE_PIX_SIZE - 1)*MINIMAP_SIZE, (this->character->getYpix() /
TILE_PIX_SIZE + 2)*MINIMAP_SIZE, MAX_ELM_HEIGHT}});
    MapTilesColor.push_back({{1.0f, 0.0f, 0.0f},
                             {1.0f, 0.0f, 0.0f},
                             {1.0f, 0.0f, 0.0f},
                             {1.0f, 0.0f, 0.0f}});
```

```
    glDisable(GL_TEXTURE_2D);
    glEnableClientState(GL_COLOR_ARRAY);

    glVertexPointer(3, GL_SHORT, 0, &MapTilesCoords.front());
    glColorPointer(3, GL_FLOAT, 0, &MapTilesColor.front());

    glDrawArrays(GL_QUADS, 0, MapTilesCoords.size()*4);

    glDisableClientState(GL_COLOR_ARRAY);
    glEnable(GL_TEXTURE_2D);
}
void Game::genMap(bool newflag)
{
    if(newflag)
    {
        this->levelMap = new LevelMap((GLuint)time(0));
        this->character = new Character(&(this->gameTime), 0, 0,
rectangles, this->textures[0]->getTexturepp(0), this->textures[0]-
>getTexturepp(144), this->textures[0]->getTexturepp(96));
    }
    else
    {
        this->levelMap->genMap();
    }
    this->character->moveToTile((FR_COLUMN - 1)*R_WIDTH + (R_WIDTH >>
1) + 1,(FR_ROW - 1)*R_HEIGHT + (R_HEIGHT >> 1) + 1);
    this->character->changeState(CONFUSE);
    this->levelCounter++;
}
void Game::gameLogic()
{
    //character moving
    this->character->changeState(this->keypress & 0xf0 ? this-
>keypress & 0xf0 : this->keypress & this->activekey & 0xf);
    this->character->updatePos(this->levelMap);
    if(this->character->testTileColision(levelMap, 3) && !this-
>character->testTileColision(levelMap, 1))
    {
        this->genMap(false);
    }
}
void Game::run()
{
    //Create SDL_openGL window
    this->WindowFlags = SDL_WINDOW_OPENGL;
    this->Window = SDL_CreateWindow("Game", 50, 50, WinWidth,
WinHeight, WindowFlags);
    assert(this->Window);
    SDL_SetWindowResizable(this->Window, SDL_TRUE);
    SDL_GLContext Context = SDL_GL_CreateContext(this->Window);
```

```
    this->Running = true;
    this->FullScreen = false;
    this->initGL();
    this->genMap(true);
    this->render();
    this->gameTime = SDL_GetTicks();
    this->nextGameLogicEvent = this->gameTime;
    //main loop
    while (Running)
    {
        this->handleEvents();
        if(!this->Pause)
        {
            GLuint frameTime = SDL_GetTicks();
            if(this->gameTime > this->nextGameLogicEvent)
            {
                this->nextGameLogicEvent = this->gameTime +
GAME_LOGIC_DELAY;
                this->gameLogic();
            }
            this->render();
            frameTime = SDL_GetTicks() - frameTime;

            if(frameTime < FRAME_TIME)
            {
                SDL_Delay(FRAME_TIME - frameTime);
                this->gameTime += FRAME_TIME;
            }

            else
            {
                std::cout << "FPS: " << 1000/frameTime << std::endl;
                this->gameTime += frameTime;
            }
        }
        else
            SDL_Delay(100);
    }
    this->clean();
    SDL_DestroyWindow(Window);
}
```

include\LevelMap.h

```
#ifndef LEVELMAP_H
#define LEVELMAP_H
#include <SDL_opengl.h>
#include <vector>
#include <ctime>
#include <stdlib.h>
#include <fstream>
```

```
#include <iostream>
#include <iomanip>
#define R_WIDTH 15//7
#define R_HEIGHT 15//7
#define R_COLUMN 11
#define R_ROW 11
#define T_WIDTH R_WIDTH*R_COLUMN
#define T_HEIGHT R_HEIGHT*R_ROW
#define MIN_ROOM_G 40
#define MAX_ROOM_G 50
#define FR_COLUMN (R_COLUMN >> 1) + 1
#define FR_ROW (R_ROW >> 1) + 1
#define DEBUG_CONSOL FALSE
#define DEBUG_FILE true
struct SubMapCoord
{
    GLushort col_off;
    GLushort row_off;
    GLushort col_N;
    GLushort row_N;
    SubMapCoord(GLshort pix_x, GLshort pix_y, GLushort pix_width,
GLushort pix_heigh, GLuint tile_size, GLuint t_width, GLuint t_height)
    {
        col_off = pix_x >= 0  ? pix_x / tile_size : 0;
        row_off = pix_y >= 0 ? pix_y / tile_size : 0;
        col_N = (pix_x + pix_width + tile_size - 1) / tile_size;
        row_N = (pix_y + pix_heigh + tile_size - 1) / tile_size;
        col_N = col_N < (t_width + 2)  ? col_N : (t_width + 1);
        row_N = row_N < (t_height + 2) ? row_N : (t_height + 1);
    }
};
struct RoomCoord
{
    GLushort col;
    GLushort row;
};
struct RoomTemplate
{
    GLubyte tiles[R_WIDTH][R_HEIGHT];
};
const float gen_p[] = {1, 0.75, 0.01, 0.005, 0};
class LevelMap
{
public:
    LevelMap(GLuint seed);
    LevelMap();
    ~LevelMap();
    void genMap();
    GLuint getSeed();
    GLubyte getTile(GLuint col, GLuint c_row);
```

```
    GLubyte getTileConectedTexture(GLuint col, GLuint c_row);
    void setSeenTile(GLuint col, GLuint c_row);
    GLubyte getSeenTile(GLuint col, GLuint c_row);
private:
    GLubyte tiles[T_HEIGHT + 2][T_WIDTH + 2];
    GLubyte tilesConectedTexture[T_HEIGHT + 2][T_WIDTH + 2];
    GLubyte seenTiles[T_HEIGHT + 2][T_WIDTH + 2];
    GLubyte tempnum[15];
    RoomTemplate* roomsTemplates[15];
    GLuint startSeed;
    GLuint seed;
    bool is_roomCreate(GLubyte neighbor);
    void inc_room(GLubyte rooms[][R_COLUMN + 2], GLuint c_col, GLuint
c_row);
    GLuint gen_Structure_from_seed(GLubyte rooms[][R_COLUMN + 2],
std::vector<RoomCoord> &noChildRooms, GLuint seed); // return room N
    GLuint gen_Structure_room_N(GLubyte rooms[][R_COLUMN + 2],
std::vector<RoomCoord> &noChildRooms, GLuint minroom, GLuint maxroom);
// return room N
    void roomConecting(GLubyte rooms[][R_COLUMN + 2]);
    void loadRoomTemp();
    void fillTile(GLubyte rooms[][R_COLUMN + 2]);
    void fillConectedTexture();
};
#endif // LEVELMAP_H
```

src\LevelMap.cpp

```
#include "LevelMap.h"
LevelMap::LevelMap(GLuint seed)
{
    this->startSeed = seed;
    this->seed = this->startSeed;
    this->loadRoomTemp();
    this->genMap();
}
LevelMap::LevelMap()
{
    this->startSeed = (GLuint)time(0);
    this->seed = this->startSeed;
    this->loadRoomTemp();
    this->genMap();
}
LevelMap::~LevelMap()
{
    for(int i = 0; i < 15; i++)
    {
        delete[] roomsTemplates[i];
    }
}
void LevelMap::genMap()
```

```
{
    GLubyte rooms[R_COLUMN + 2][R_ROW + 2];
    std::vector<RoomCoord> noChildRooms;
    //next level seed
    this->seed *= this->seed + 1;
    //std::cout << this->seed << std::endl;
    //gen structure
    GLuint roomN = this->gen_Structure_room_N(rooms, noChildRooms,
MIN_ROOM_G, MAX_ROOM_G);
    this->roomConecting(rooms);
    if(DEBUG_CONSOL)
    {
        std::cout << roomN << std::endl;
        for(int i = 1; i < R_ROW + 1; i++)
        {
            for(int j = 1; j < R_COLUMN + 1; j++)
                std::cout << std::setfill(' ') << std::setw(2) <<
(rooms[i][j]?"H":" ");
            std::cout << std::endl;
        }
    }
    //fill rooms
    this->fillTile(rooms);
    //place special room
    RoomCoord exitroom = noChildRooms[rand() % noChildRooms.size()];
    GLuint pit_x = R_WIDTH*(exitroom.col - 1) + (R_WIDTH >> 1);
    GLuint pit_y = R_HEIGHT*(exitroom.row - 1)  + (R_HEIGHT>> 1);
    for(int i = 0; i < 3; i++)
        for(int j = 0; j < 3; j++)
            this->tiles[pit_y + i][pit_x + j] = 3; //exit pit
    this->fillConectedTexture();

    if(DEBUG_FILE)
    {
        std::ofstream mapfile("map.txt", std::ios::out);
        const char contex4[] = {254,186, 205, 200, 186, 186, 201,
204, 205,  188,  205,  202,  187, 185,  203, 206};
        for(int i = 0; i < T_HEIGHT + 2; i++)
        {
            for(int j = 0; j < T_WIDTH + 2; j++)
                if(tiles[i][j] == 0)
                    mapfile << " " << " ";
                else if (tiles[i][j] == 1)
                    mapfile << "." << " ";
                else if (tiles[i][j] == 2)
                    mapfile << "H" << " ";
                else if (tiles[i][j] == 3)
                    mapfile << "E" << " ";
                else
                    mapfile << "e" << " ";
```

```
            mapfile << std::endl;
        }
        for(int i = 0; i < T_HEIGHT + 2; i++)
        {
            for(int j = 0; j < T_WIDTH + 2; j++)
                if(tilesConectedTexture[i][j] != 255)
                    mapfile << std::setfill(' ') << std::setw(3) <<
(short)(tilesConectedTexture[i][j]) << " ";
                else
                    mapfile << std::setfill(' ') << std::setw(3) << "
" << " ";
            mapfile << std::endl;
        }
        mapfile.close();
    }
    for(int i = 0; i < T_HEIGHT + 2; i++)
    {
        for(int j = 0; j < T_WIDTH + 2; j++)
        {
            this->seenTiles[i][j] = 0;
        }
    }
}
GLuint LevelMap::getSeed()
{
    return this->startSeed;
}
bool LevelMap::is_roomCreate(GLubyte neighbor)
{
    if(neighbor > 4)
        return false;
    if((float)rand()/RAND_MAX < gen_p[neighbor])
    {
        return true;
    }
    return false;
}
void LevelMap::inc_room(GLubyte rooms[][R_COLUMN + 2], GLuint c_col,
GLuint c_row)
{
    rooms[c_row][c_col] = 5;
    rooms[c_row + 1][c_col]++;
    rooms[c_row - 1][c_col]++;
    rooms[c_row][c_col + 1]++;
    rooms[c_row][c_col - 1]++;
}
GLuint LevelMap::gen_Structure_from_seed(GLubyte rooms[][R_COLUMN +
2], std::vector<RoomCoord> &noChildRooms, GLuint seed)
{
    //feel rooms
```

```
    for(int i = 0; i < R_ROW + 2; i++)
    {
        for(int j = 0; j < R_COLUMN + 2; j++)
        {
            rooms[i][j] = 0;
        }
    }
    srand(seed);
    //first room placing
    std::vector<RoomCoord> active_room;
    active_room.push_back({FR_COLUMN, FR_ROW});
    this->inc_room(rooms, active_room[0].col, active_room[0].row);

    //room placing
    GLuint roomN = 1;
    while(!active_room.empty())
    {
        GLuint active = rand() % active_room.size();
        GLubyte nr_col = active_room[active].col;
        GLubyte nr_row = active_room[active].row - 1;

        bool nochildflag = true;
        if(active_room[active].row != 1 && this-
>is_roomCreate(rooms[nr_row][nr_col]))
        {
            this->inc_room(rooms, nr_col, nr_row);
            active_room.push_back({nr_col, nr_row});
            nochildflag = false;
            roomN++;
        }
        nr_row = active_room[active].row + 1;
        if(active_room[active].row != R_ROW && this-
>is_roomCreate(rooms[nr_row][nr_col]))
        {
            this->inc_room(rooms, nr_col, nr_row);
            active_room.push_back({nr_col, nr_row});
            nochildflag = false;
            roomN++;
        }
        nr_row = active_room[active].row;
        nr_col = active_room[active].col - 1;
        if(active_room[active].col != 1 && this-
>is_roomCreate(rooms[nr_row][nr_col]))
        {
            this->inc_room(rooms, nr_col, nr_row);
            active_room.push_back({nr_col, nr_row});
            nochildflag = false;
            roomN++;
        }
        nr_col = active_room[active].col + 1;
```

```
        if(active_room[active].col != R_COLUMN && this-
>is_roomCreate(rooms[nr_row][nr_col]))
        {
            this->inc_room(rooms, nr_col, nr_row);
            active_room.push_back({nr_col, nr_row});
            nochildflag = false;
            roomN++;
        }
        if(nochildflag)
            noChildRooms.push_back(active_room[active]);

        active_room[active] = active_room.back();
        active_room.pop_back();

    }
    return roomN;
}
GLuint LevelMap::gen_Structure_room_N(GLubyte rooms[][R_COLUMN + 2],
std::vector<RoomCoord> &noChildRooms, GLuint minroom, GLuint maxroom)
{
    GLuint cur_seed = this->seed;
    GLuint best_seed = this->seed;
    GLuint minErr = INFINITE;
    GLuint roomN;
    for(int i = 0; i < 1000; i++)
    {
        noChildRooms.clear();
        cur_seed += this->seed;
        roomN = this->gen_Structure_from_seed(rooms, noChildRooms,
cur_seed);
        if(roomN > minroom && roomN < maxroom) //&&
!noChildRooms.empty())
        {
            return roomN;
        }
        else
        {
            if(roomN > minroom)
            {

                if(minErr > roomN - maxroom)
                {
                    minErr = roomN - maxroom;
                    best_seed = cur_seed;
                }
            }
            else
            {
                if(minErr > minroom - roomN)
                {
```

```
                    minErr = minroom - roomN;
                    best_seed = cur_seed;
                }
            }
        }
    }
    std::cout << "err" << minErr << std::endl;
    noChildRooms.clear();
    roomN = this->gen_Structure_from_seed(rooms, noChildRooms,
best_seed);
    return roomN;
}

void LevelMap::roomConecting(GLubyte rooms[][R_COLUMN + 2])
{
    //room finding
    for(int i = 0; i < R_ROW + 2; i++)
    {
        for(int j = 0; j < R_COLUMN + 2 ; j++)
            if(rooms[i][j] > 4)
                rooms[i][j] = 1;
            else
                rooms[i][j] = 0;
    }
    //connecting room
    for(int i = 1; i < R_ROW + 1; i++)
    {
        for(int j = 1; j < R_COLUMN + 1 ; j++)
            if(rooms[i][j] == 1)
                rooms[i][j] =   ((rooms[i-1][j] && 1) +
((rooms[i][j+1] && 1) << 1) +
                                 ((rooms[i + 1][j] && 1) << 2) +
((rooms[i][j-1] && 1) << 3));
    }
}
void LevelMap::loadRoomTemp()
{
    std::ifstream roomsfile("rooms.lrm",
std::ios::in|std::ios::binary);
    for(int i = 0; i < 15; i++)
    {
        roomsfile.read((char*)(&this->tempnum[i]), 1);
        this->roomsTemplates[i] = new RoomTemplate[this->tempnum[i]];
        //for(int j = 0; j < this->tempnum[i]; j++)
        roomsfile.read((char*)(roomsTemplates[i]),
tempnum[i]*R_WIDTH*R_HEIGHT);
    }
    roomsfile.close();
}
void LevelMap::fillTile(GLubyte rooms[][R_COLUMN + 2])
```

```
{
    srand(this->seed);
    for(int i = 1; i < R_ROW + 1; i++)
    {
        for(int j = 1; j < R_COLUMN + 1 ; j++)
        {
            int col_off = 1 + (j - 1)*R_WIDTH;
            int row_off = 1 + (i - 1)*R_HEIGHT;
            if(rooms[i][j] != 0)
            {
                //std::cout << row_off << " " << col_off << std::endl;
                RoomTemplate* room = roomsTemplates[rooms[i][j] - 1] +
rand() % this->tempnum[rooms[i][j] - 1];
                for(int ti = 0; ti < R_HEIGHT; ti++)
                {
                    for(int tj = 0; tj < R_WIDTH; tj++)
                    {
                        this->tiles[ti + row_off][tj + col_off] =
room->tiles[ti][tj];
                    }
                }
            }
            else
            {
                for(int ti = 0; ti < R_HEIGHT; ti++)
                {
                    for(int tj = 0; tj < R_WIDTH; tj++)
                    {
                        this->tiles[ti + row_off][tj + col_off] = 2;
                    }
                }
            }
        }
    }
    //border fill
    for(int i = 0; i < T_WIDTH + 2; i++)
    {
        this->tiles[0][i] = 2;
        this->tiles[T_HEIGHT + 1][i] = 2;
    }
    for(int i = 0; i < T_WIDTH + 2; i++)
    {
        this->tiles[i][0] = 2;
        this->tiles[i][T_WIDTH + 1] = 2;
    }
}
void LevelMap::fillConectedTexture() //H mirror
{
    for(int i = 1; i < T_HEIGHT + 1; i++)
    {
```

```
        for(int j = 1; j < T_WIDTH + 1 ; j++)
            if(this->tiles[i][j] != 0)
                this->tilesConectedTexture[i][j] =  ((tiles[i][j] ==
tiles[i+1][j]) << 4) + ((tiles[i][j] == tiles[i][j+1]) << 5) +
((tiles[i][j] == tiles[i-1][j]) << 6) + ((tiles[i][j] == tiles[i][j-
1]) << 7) +
((tiles[i][j] == tiles[i+1][j+1]))      + ((tiles[i][j] == tiles[i-
1][j+1]) << 1) +
((tiles[i][j] == tiles[i-1][j-1]) << 2) + ((tiles[i][j] ==
tiles[i+1][j-1]) << 3);
    }
    //border fill
    for(int i = 0; i < T_WIDTH + 2; i++)
    {
        this->tilesConectedTexture[0][i] = 0xff;
        this->tilesConectedTexture[T_HEIGHT + 1][i] = 0xff;
    }
    for(int i = 0; i < T_WIDTH + 2; i++)
    {
        this->tilesConectedTexture[i][0] = 0xff;
        this->tilesConectedTexture[i][T_WIDTH + 1] = 0xff;
    }
}
GLubyte LevelMap::getTile(GLuint col, GLuint row)
{
    return this->tiles[row][col];
}

GLubyte LevelMap::getTileConectedTexture(GLuint col, GLuint row)
{
    return this->tilesConectedTexture[row][col];
}

void LevelMap::setSeenTile(GLuint col, GLuint row)
{
    this->seenTiles[row][col] = 1;
}

GLubyte LevelMap::getSeenTile(GLuint col, GLuint row)
{
    return this->seenTiles[row][col];
}
```